Spring 7-1-2013

DLCM Programmer Interface

Tilden C. May
Dakota State University

DLCM PROGRAMMER INTERFACE

A graduate project submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Master of Science

in

Information Systems

July, 2013

By
Tilden C. May

Project Committee:

Ronghua Shan
Christopher Olson

PROJECT APPROVAL FORM

We certify that we have read this project and that, in our opinion, it is satisfactory in scope and quality as a project for the degree of Master of Science in Information Systems.

Student Name: Tilden May

Master's Project Title: **DLCM Programmer Interface**

Faculty supervisor: Ronghua Shan Date: 8/16/13

Committee member: Chris Kloon Date: 8/16/2013

Committee member: Rick Christoph Date: 8/17/13

ABSTRACT

This project focuses on an information systems solution that serves to provide a method for flashing microprocessors on automotive control units in a high volume mass production setting. In order to accomplish this, an integration application was developed to mediate between the control mechanisms of an assembly machine and the handling instructions of the EEPROM flashing module to program and process the electronic product being manufactured. An additional attribute of the system is to preserve the processing data from each product being manufactured into a systems database. A third element of the information systems project is the deployment of a website that provides production and quality support staff an avenue for retrieving and assessing the flashing results for a given product.

During the scope of this project, focus centered on the primary SDLC phases and methods of developing an information systems project. These included initial planning, requirements analysis, design and development, and finally implementation and maintenance stages. Artifacts and deliverables are included that reflect the progression throughout the course of this software development life cycle.

To solve the problem of creating an automated high capacity method for flashing the control units, a VB.Net client application utilizing API's and multi-threading serves as the primary interface on a custom engineered assembly machine. Prior to this solution, the method involved manually connecting and interacting with each product which was inefficient and allowed for a high possibility for error and lack of quality control. This was not an acceptable means for creating a mass production process. An SQL relational database is used for storing traceability data from each product transaction that is processed through the machine. The client application performs the database updates to the server. The system website is launched using JSP and servlets technology and runs on the application server GlassFish.

This information systems solution was developed and implemented in parallel with an entire new assembly line to produce automotive drive line control modules for Chrysler vehicles. Under the constraints of an aggressive deadline and customer requirements, the

installation and system objectives were met and implemented on time and all tiers of the system function as intended and specified.

DECLARATION

I hereby certify that this project constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the project describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Tilden C. May

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Background of the Problem

In order to provide an understanding of the scope of this project, it is first imperative to establish a proper background of the working environment and setting that the project will be implemented in. The firm setting is Hitachi Automotive Systems which provides Tier 1 and Tier 2 electronic automotive parts to a variety of customers including Nissan, Chrysler, Toyota, GM, and many others. Primary responsibilities include to project manage from an engineering standpoint, the procurement, technical cohesiveness, functionality, and effective installation of all the equipment and machinery needed to form a mass production assembly line. These are typically highly automated and may consist of fifty to a hundred or more individual pieces of equipment. The variation in equipment specifications and quantities typically depends on the specific design of the product being produced and what special requirements are necessary. For many of the processes there are a host of industry leading vendors who specialize in that particular manufacturing step and provide an off-the-shelf solution. However there is typically some portion of the manufacturing process that, due to the unique design features of the product, requires a custom level system or machine. Often the phase from project kick-off to full installation will span in the range of one year or more. During this time, all the technical details for each process have to be determined; quotations and procurement decisions made, equipment designed, developed, and delivered, and ultimately full installation and integration will take place. Much like the phases of SDLC, once all systems are in place and the actual production phase launches, there are continual iterative needs for system maintenance, enhancements, and improvements throughout the project life cycle.

This capstone project focuses on a specific subset of the larger complete assembly line implementation. That project consisted of a $4 million capital investment to establish a new production line to produce electronic control modules called 'DLCM'. This acronym stands for "drive line control module" and the customer for these units is American Axle, who then

supplies to Chrysler. The DLCM controls the vehicle drive-train or power-train system. (Chaikin, 2004) The production line process begins with a single blank or raw printed circuit board "pcb", and concludes with a fully assembled and functional module that is ready to be installed on a new car at the customer's assembly line process. The illustration below shows an exploded view of a completed unit.

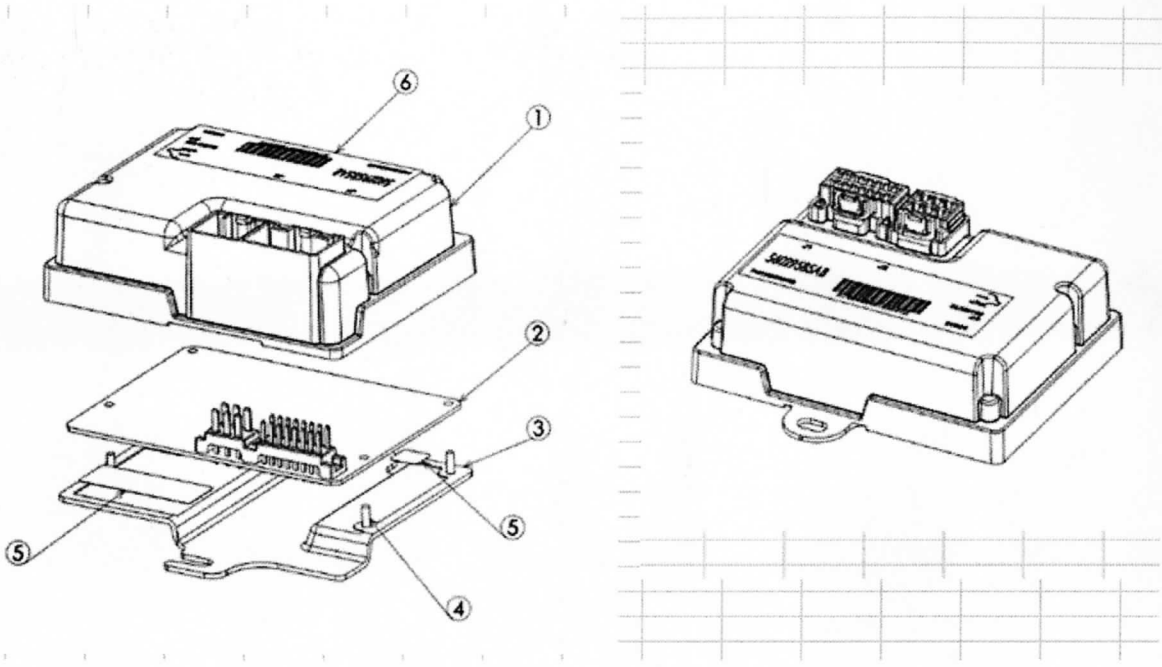


Figure 1: DLCM Module Depiction

An integral step during the production of this module is the process of flashing the onboard microprocessor. Without this, the unit is a dead circuit board without any functionality. This critical step is the basis for this information systems project. Figure 1 below shows a CAD layout of the DLCM production line. This is a process layout of the equipment required for mass production and gives you an idea of the process flow and overall size and scale. Depicted in blue is the microprocessor flashing station.

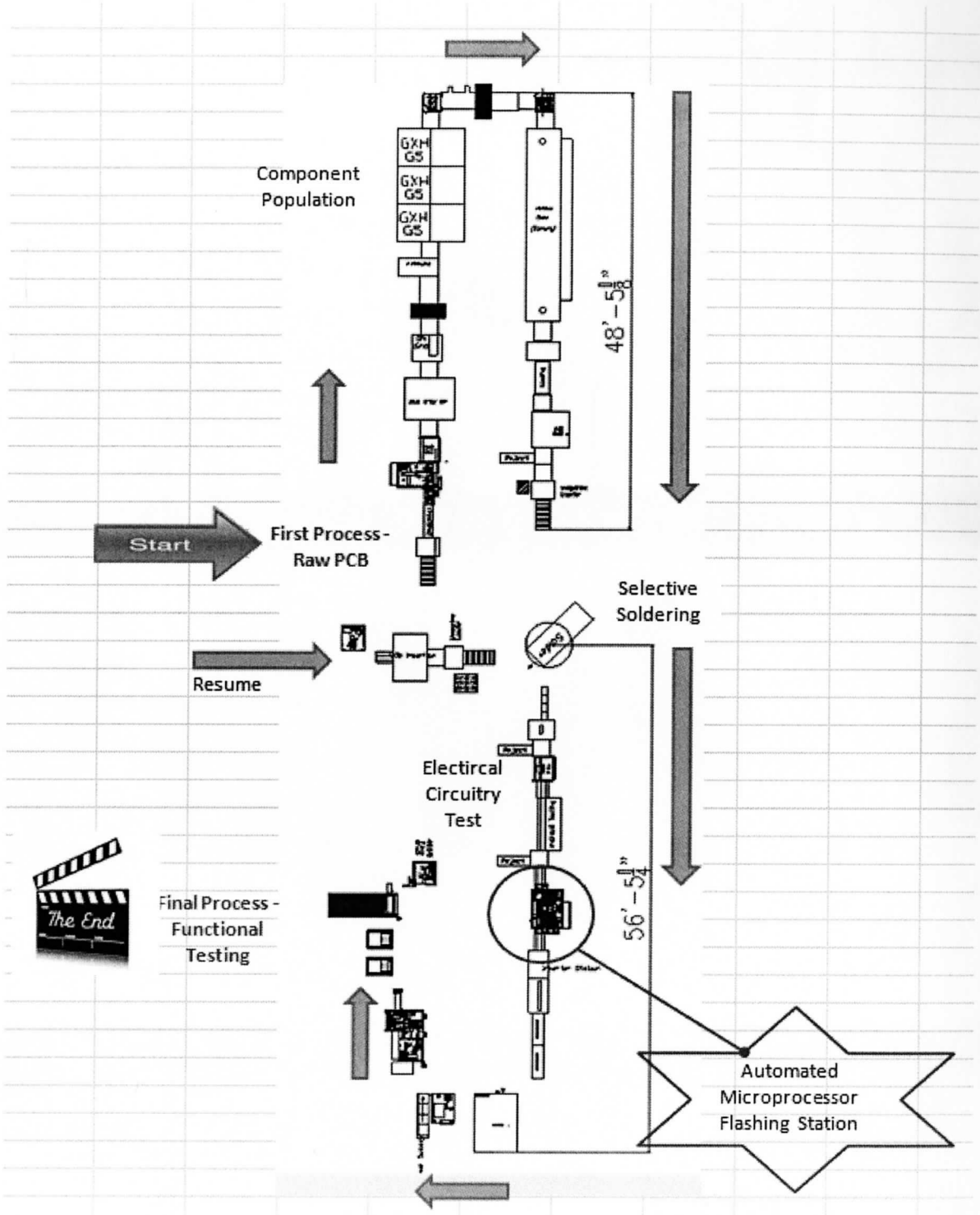


Figure 2: FA5 Production Line

Statement of the Problem

As alluded to previously, many assembly processes can be solved by systems offered from a variety of vendors offering off-the-shelf solutions. The microprocessor flashing process by its very nature has some unique characteristics that typically put it into a class that tends to necessitate some level of customization. This is due to the vast variance in circuit board design with pcb size, programming pad locations and geometry, microprocessor type selection, and the proprietary nature of the programming file itself, just to name a few. For these reasons, it was necessary in this case, to develop a customized method to perform the micro flashing and do so in a manner that achieved a high volume / mass production capable process. It is important to keep in mind that one of the key objectives is to produce a system that is mass production capable. The tools and utilities required to simply flash program a module in a lab environment are already available and are used during the product design and testing phase. The steps outlined in figure 3 illustrate how this is done. They include placing the product onto the programming fixture, securing and connecting the appropriate electrical terminals, interfacing with the Cyclone Pro flashing tool, and finally initiating a programming sequence.

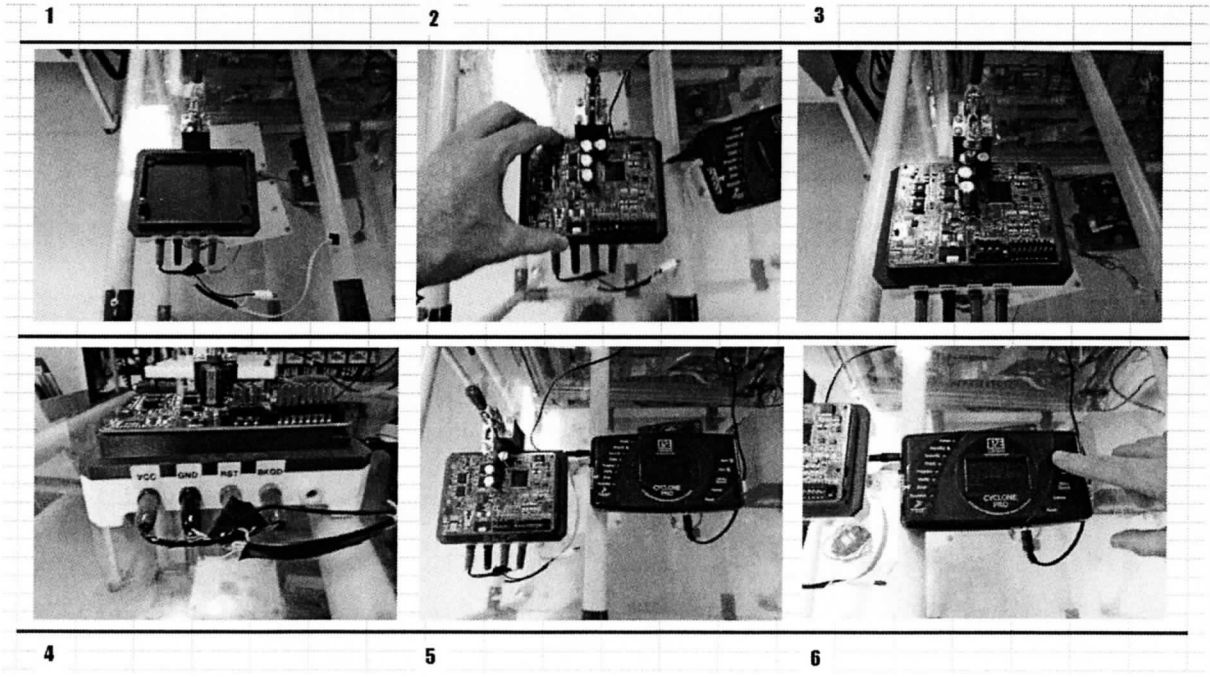


Figure 3: Manual Flashing Process

While this is a simple and effective method for programming in a stand-alone scenario suitable for a lab setting, it poses many issues when viewed as a solution for a mass production environment. These are detailed below.

Table 1: Manual Flashing Cons

Issue	Comments
Manual Handling	Quality Concerns, unnecessary handling introduces damage potential
Labor Intensive	Requires labor cost to perform, can be multiplied based on volume / work shifts needed
Procedure Accuracy	Chance for mistakes in manually selecting programming files, and initiating programming sequence
Results Handling	Chance to mishandle product if part failed to program correctly; no adequate interface or user feedback
Traceability Tracking	No data tracking incorporated

Other factors in determining the best approach and the level of automation necessary for this process is to consider the volumes that have to be produced and the cycle time requirements that have to be achieved. These determinations are a byproduct of capacity calculations that consider; product volume, number of shifts, working minutes, number of staff, worker fatigue, and anticipated system uptime. In this case the cycle time requirement is 30 seconds. The DLCM product, while still in a raw pcb format, is supplied in pairs of two. These later become separated by a router machine before they are assembled as one unit. The programming time based on lab trials is approximately two minutes to erase, format, program, and verify the EEPROM. With this basic information, it can be ascertained that the automated programming system needs to be capable of flashing four PCB's at one time. (*Programming Time* -> 120 seconds / 4 *units* = 30 *seconds*). The chart below in figure 4 illustrates the product volume forecasted through 2016. It is segmented by bi-annual budget periods.

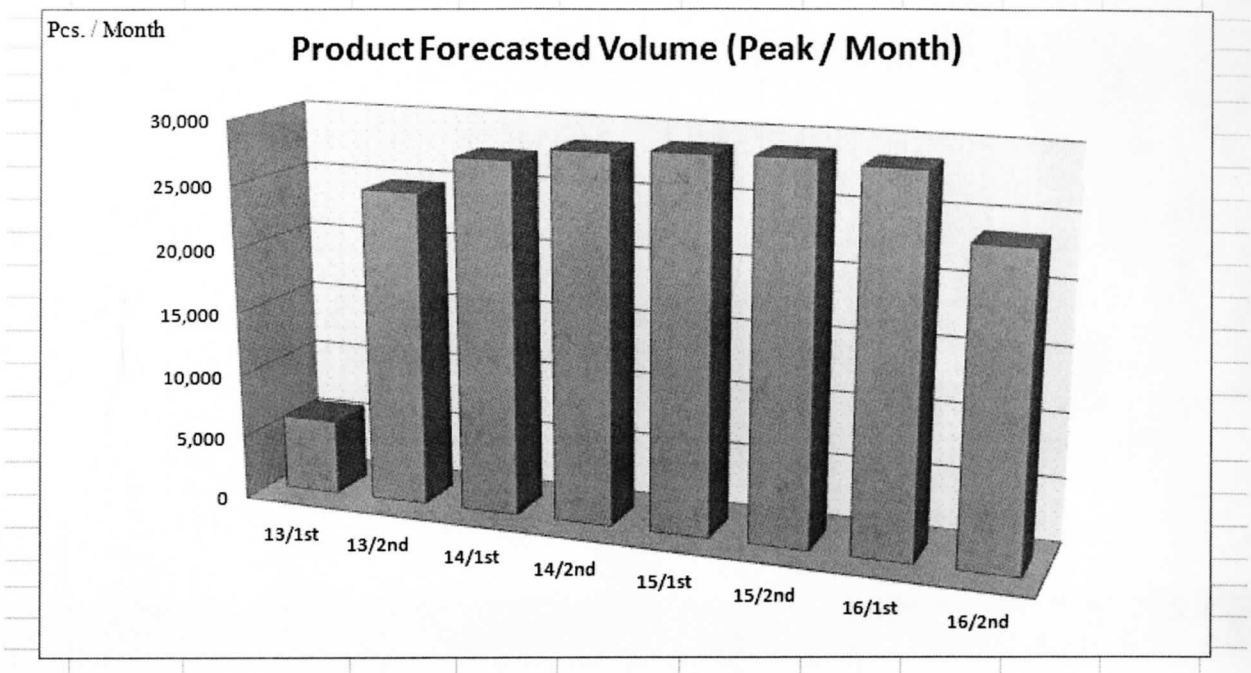


Figure 4: Product Volume Chart

In summary, the lab environment method for flashing the microprocessor is not suitable for high volume mass production. An automated system needs to be designed and developed that is capable of flashing a minimum of four modules at one time. The system should address all of the concerns and shortcomings that the manual procedure presents.

Objectives of the Project

With a general background established and an overview of the problem cited, the objective details that the system should meet and the deliverables expected can be outlined. An automated controls-based machine will be developed to handle the programming sequence. It will handle in an automated manner, the product fixturing, electrical connectivity, and travel / throughput of the DLCM product.

This system will be comprised of three main components. The primary one is a client Windows application that will serve as an integration interface that will have two main purposes: (1) API control of the Cyclone Pro Flashing Units. These are the flashing tools that execute the erasing / formatting / programming instructions onto the product microprocessor. (Cyclone Pro, 2013) (2) API interaction with the machine side controls via the HMI, or 'Human Machine Interface'. (Definition of HMI) The HMI handles the plc controls of the machine, thus the logic to activate all conveying, robotics, and pneumatics. The logic exchange between the client application and the HMI include start and stop conditions of the programming sequence, conclusive results of either success or failure for each programming cycle, and handling the individual barcode tracking for each product for traceability purposes.

The interface should allow for easy user interaction, including model change-over selection, real-time status feedback for all stations, and the ability to run in an automated mode as well to individually interact with the Cyclone Pro Units for debugging purposes.

The second component of the system is to develop and implement an SQL database that will store the programming events for historical traceability data. The database will reside on a server, utilize stored procedures, and capture the relevant data items associated with each programming sequence cycle.

The third major component of the system is the development of an intranet website that will allow production support staff to have the capability to easily query a specific product to gather all the results from the microprocessor flashing station. The website will access the database and can be a valuable tool for quality investigations and as a method to verify the outcome of the programming process.

The table below summarizes the three major system components.

Table 2: System Objectives

System Objectives	
Component 1	Windows Application – API Integration with Cyclone Pro and ProFace HMI
Component 2	Microsoft SQL Database – Record Microprocessor Flashing Events
Component 3	Java Servlets Website – Provide Web Query of Programmer Information

Within the three major components outlined above, the deliverables detailed below will comprise the foundation of the project and are included in this report.

System deliverables

Planning Phase

- Feasibility Analysis
  - Economic Feasibility
  - Technical Feasibility
  - Organizational Feasibility
- Work Breakdown Schedule
- Gantt Chart

Analysis Phase

- Requirements Definition
  - Nonfunctional
  - Functional

Design Phase

- Activity Diagram
- Database Diagram
- Network Diagram

- Hardware / Software Specification
- Windows Integration Application Source Code
- Website Source Code
- SQL Stored Procedures
- User Technical Manual

CHAPTER 2

LITERATURE REVIEW

One of the crucial development elements of the integration application that needed to be accounted for was the importance of handling multiple threads to accommodate concurrent tasks. Research was conducted to answer the questions of what a thread is and why and when it is necessary to invoke multiple threads within an application.

(Meisel, 2007) discussed the evolving use of software multithreading to coincide with the enhancements in processing architecture and the development of multi-core processing. The author provides an overview of the concept of multithreading describing how threads reside within a process and the threads can perform different tasks while having access to the same functions and data. He points out that as a developer, the work involved in creating the thread codes is minimal, however the assurance that the threads are managed and allow for proper synchronization is both critical and more time consuming. Thread management can be described as controlling thread sequences if the threads need access to the same memory allocation. One method for controlling this is to use locks which allow a thread to lock out all other threads from accessing specific resources until the locking thread is finished. Secondary threads attempting to access the resources will wait and resume once the lock has been lifted. This concept of defining the execution order among multiple threads is noted by the author to be synchronization. The most common form of synchronization is mutual exclusion where a thread will block out a critical section of code from other threads until the thread has competed. The author concludes that the continued progression towards multi-core processing makes the need to understand and master multithreading concepts an important caveat for software developers.

(Woodring & Cohen, 1997) examined the benefits of working with multiple threads within an application. One is an ability to achieve parallelization. In a single thread instance independent tasks are completed serially and the total processing time is the sum of the duration time that each task entails. With the ability to perform these same tasks in parallel,

the total processing time is equal to the task with the longest duration time only. Additional benefits proposed include; simplified programming design, increased robustness for performing critical tasks, maintaining user responsiveness, and improved usage efficiency of the CPU and operating system. Along with discussing the benefits of multithreading, the authors also point out reasons not to utilize multiple threads. The first is to always have a good reason to do so; using additional threads involves added design, implementation, and debugging time to the development process so ensure that the tasks necessitate it. Secondly, consider the overhead applied to the operating system and whether or not this outweighs the benefit of the additional thread. In cases where the thread would be called at very frequent intervals and the work being done is minimal it may be detrimental to not only the application but the workstation as a whole. The authors summarize that it is vital for programmers to understand the effect on execution in a scenario where multiple threads have access to the same functions.

(Morrison, 2005) discussed the importance of understanding how multiple threads share and access memory and the consequences that can arise from this. A condition known as a race can occur when differing threads simultaneously access a common memory location causing incorrect computation of the data structure. A common means of preventing a race is to instill locks that prevent an outside thread from accessing the memory while it is in use with an existing thread. Locks go by many names; monitor, critical section, mutex, and binary semaphore. Regardless of the moniker, the principal functionality of the lock is to provide an Enter and Exit method for accessing a portion of code or section of memory. Once a thread calls the lock Enter method any attempt by an outside thread to access the same code will be blocked. Not until the Exit method is raised can the requesting thread proceed to execute the same code or gain memory access. It is noted that the ultimate goal should be to protect memory location execution as opposed to sections of code. If there are multiple paths to a set of common memory then instilling a lock in one instance does not guarantee that an outside thread is not accessing the same memory through an alternate path simultaneously. This needs to be accounted for during the program design. Care should be taken to use as few locks as possible however. Locks incur overhead by using special instructions to evaluate whether a lock is already in use or not and whether competing processors are attempting to use the same lock. Another risk for using a high number of locks is a phenomenon known as

deadlock can occur. In this scenario threads try to access multiple locks in which the competing thread already owns. Both become in a state of waiting where neither can proceed because they are depending on the other to complete. In this article, the author summarizes that in multithreaded programs, it is important to protect memory access by the inclusion of locks. It is equally important to minimize the overall usage of locks and structure the program in a manner that allows the lock to seclude the memory from other competing portions of code. Successfully implemented, the application can perform free from races and deadlock conditions.

In examining different perspectives on the usage of threads within application programming, a clearer understanding of what a thread is and why we should use it comes to light. In its basic form, a thread exists within a process and allows the execution of tasks to be performed in parallel as opposed to sequentially. In conjunction with hardware advancements in multi-core processing the usage of multiple threads provides improved benefits such as computing efficiency, user interface responsiveness, and a robust method for performing system critical tasks. Although there are clear conditions in which the usage of additional threads are recommended it is imperative to have a sound understanding of the risks and pitfalls involved if the design and structure of the program is not adequately established. The proper use of locks to impose mutual exclusion can help the developer eliminate potential races and deadlock conditions which can be difficult to debug and resolve. It is a given that advancements in processors will continue to expand the possibilities of what software applications and systems are capable of providing. As a developer, mastering the proper usage of multithreading, is an important skill that should be a part of one's programming arsenal.

CHAPTER 3

SYSTEM: PLANNING AND ANALYSIS

Economic Feasibility

As briefly mentioned before, the economic feasibility for this project is actually a subset of a much larger project; a $4 million investment was approved to procure and install a new automated production assembly line to produce electronic control modules for the customer, American Axle. The product specifically is a 'drive line control module', and will be referred throughout this document as DLCM. There are two applications for the controller, a single and dual speed version. The projected volume has a potential peak of 447,000 pieces per year though the year 2018. This project focuses directly on the process of developing a system to flash the microprocessor on the DLCM in a high-speed production environment. The target allowance for the inline flashing station project is $125,000 and is based upon comparison of historical data on similar projects and types of equipment.

Although there are many vendors who excel within the industry at certain manufacturing processes, this particular process will be designed, developed, and implemented by the firm's internal engineering staff. There are many reasons for this but primarily it is due to the custom nature of flashing microprocessors versus a commodity type process. In addition, Hitachi has the technical capability to build machine automation and the expertise to integrate with peripheral systems. This in-house approach will aid in controlling the project cost with a mindset of completing within the allotted budget.

The costs for developing the system will include; design time, hardware procurement, software licensing, fabrication, installation, programming, debugging, and utilities. The following sections provide the details of the economic feasibility study.

Table 3: Cost vs. Benefit Analysis

Development Costs	Operational Costs
Core Team Salaries	Software Maintenance / System Enhancement
Hardware / Software Purchases	Hardware Maintenance - Repair / Replace
3D Modeling Design / Development	
Tool / Die Components	
Equipment Fabrication	
Installation / Debugging	

Tangible Benefits	Intangible Benefits
Increased Sales / New Product Launch	High Quality / Repeatability Process
Automated Flashing Method	Maintain Customer Relationship
Achieve Required Cycle Time	Team Member Friendly Operation Interface
Product Traceability Data	Platform For Future Equipment Design

Table 4: Development Cost Breakdown

Item Description	Unit Cost	Labor hours	Quantity	Total Cost
Software Development	----------	240		$18,000.00
Client PC	$845.64		1	$845.64
Cyclone Pro In-Circuit Programmer	$587.50		4	$2,350.00
HMI (ProFace AGP3400)	$2,027.50		1	$2,027.50
PLC (Including I/O Modules / Ethernet)	$9,233.91		1	$9,233.91
Electrical Control Components / Wiring	----------			$27,949.71
Mechanical Hardware (Tooling / Frame)	----------			$12,722.53
Database Server	$9,730.76		1	$9,730.76
MS SQL 2008 WorkGroup (server + 10 CAL's)	$3,749.51		1	$3,749.51
Cognex DataMan Barcode Reader	$3,035.00		1	$3,035.00
Labor (Design / Fabrication / Programming)	----------	400		$30,000
Total Development Costs:		640		$119,644.56

*Note: Projected Labor Rate = $ 75.00 / Hr

Cost Benefit Analysis

The figure below is a snapshot of the 'Return On Investment' calculation of the entire project as a whole. The investment is just under $4 million with an ROI of 14% and a payback period of just over two years.

6.) T&N: Only for Investment Jobs
(Key Amounts in Yellow Shaded Cells)

Improvement ROI & PayBack Period		Target	Difference
Investment	-		
Cost Savings (Before Tax)/Month	-		
Depreciation per Month	-		
HQ Total Additional Cost Factor	1.22		
Tax Rate	*20.00%*		
Payback Period (in months) (N)	*0*	*24.0*	*(24.0)*
Return on Investment (T)	*0*	*30.00%*	*-30.00%*

Sales Increase ROI and Payback Periods		Target	Difference
Investment (K$)	3,998		
Increase in Sales (K$/Month)	1,214		
Monthly Income Before Tax (K$/Month)	123		
Monthly Gross Profit After Tax (K$/Month)	98		
Sales additional value factor	4.5		
Payback Period (in months) (N)	*24.2*	*24.0*	*0.2*
Return on Investment (T)	*14.27%*	*10.00%*	*4.27%*

Figure 5: Investment / Return Calculation

In the following figure, a breakdown of the total cost by assembly process is provided. The automated microprocessor flashing station is highlighted in yellow with a budgeted cost of $125,000.

ACCT USE	ITEM NO.	SUPPLIER	DESCRIPTION	AMOUNT
	1	~~Universal~~	PCB Loader	$ 15,000.00
	2	~~(HIAMS)AM-HK~~	Laser Etch	$ 115,000.00
	3	~~Universal~~	Conveyor	$ 4,000.00
	4	~~DEK~~	Solder Paste Screen Printer	$ 160,000.00
	5	~~Universal~~	Conveyor	$ 4,000.00
	6	~~Omron-CKD~~	Paste Inspection	$ 155,000.00
	7	~~Universal~~	Conveyor	$ 4,000.00
	8	~~Asymtek~~	Adhesive Dispense	$ 135,000.00
	9	~~Universal~~	Conveyor	$ 4,000.00
	10	~~HHT~~	Component Placement Mounter (Sigma)	$ 340,000.00
	11	~~HHT~~	Component Placement Mounter (Sigma)	$ 340,000.00
	12	~~HHT~~	IC - Mounter (Sigma)	$ 400,000.00
	13	~~Universal~~	Conveyor	$ 4,000.00
	14	~~Tamura~~	Reflow (Pb-Free)	$ 225,000.00
	15	~~Universal~~	FIFO Accumulator	$ 27,000.00
	16	~~Omron~~	WIN II AOI	$ 256,000.00
	17	~~Universal~~	NG turn / Conveyor	$ 25,000.00
	18	~~Universal~~	Magazine Buffer	$ 24,000.00
	19	~~(HIAMS)AM-HK~~	Connector Insertion	$ 20,000.00
	20	~~Pillarhouse~~	Selective Solder	$ 113,000.00
	21	~~Yamaha~~	Connector Pin AOI	$ 174,000.00
	22	~~Accelerate~~	In-Circuit Test	$ 175,000.00
	23	(HIAMS)AM-HK	Microprocessor Flashing Station	$ 125,000.00
	24	~~Asys~~	PCB de-paneling	$ 124,000.00
	25	~~(HIAMS)AM-HK~~	Ionized Air Blow	$ 27,000.00
	26	~~(HIAMS)AM-HK~~	Thermal Grease Dispense	$ 145,000.00
	27	~~(HIAMS)AM-HK~~	PCB Place/Cover Place/Screw Insert	$ 25,000.00
	28	~~(HIAMS)AM-HK~~	Label Placement	$ 12,000.00
	29	~~(HIAMS)AM-HK~~	Cure Oven	$ 225,000.00
	30	~~(HIAMS)AM-HK~~	Final Functional Test (Qty: 2)	$ 300,000.00
	31	~~(HIAMS)AM-HK~~	Final Inspection	$ 125,000.00
	32	~~(HIAMS)AM-HK~~	Smart Bin	$ 20,000.00
	33	~~(HIAMS)AM-HK~~	Facilities / Traceability	$ 151,140.00
			TOTAL:	$ 3,998,140.00

Figure 6: Equipment Item Detail

Technical Feasibility

This inline microprocessor flashing station is a sub process of a complete automated manufacturing line that has to be procured, developed, and installed within a tight schedule. One of the initial determinations that needed to be resolved was whether this project will be outsourced or completed internally by the firm. Within the organization, an experienced group of engineers exist who develop machine automation and have built similar pieces of equipment in the past. The fundamental operation of this process is to flash program Freescale Microprocessors on electronic control module circuit boards. The core team has experience working with the Cyclone Pro units that will perform the flashing procedure. The role of the software application will be to integrate with the machine side control to handle product flow and convey the status of the programming actions including the starting sequence and providing the results. In addition, the application will integrate with the Cyclone Pro Units to perform initialization, image file transferring, and initiate the programming sequence.

The machine side application will be developed using VB.Net 2010 within Visual Studio IDE. A database for storing the results will be created using Microsoft SQL Server. An intranet website using Java servlets and running on GlassFish server will provide users with data retrieval access. Unlike some other processes which are more of a commodity and have many industry vendors to choose from, the flashing station is very customizable due to it being dependent upon the circuit design, pcb size, and the selection type of microprocessor being used. Given this facet along with the organization's expertise, capability, and experience, a recommendation was concluded that the internal group shall proceed with the task of completing this project.

Organizational Feasibility

This project is an integral process that is part of a complete capital investment of a new automated manufacturing assembly line to produce drive line control modules for American Axle and Chrysler. The product volume forecasts as high as 447,000 pieces per year through 2013 to the year 2018. The development of this automated microprocessor flashing system aligns with the strategic focus of the firm, to be a world class leader in producing automotive electronic systems, with the highest regard for quality, technical innovation, and customer satisfaction.

The economic feasibility report has been accepted and approved by corporate management. The technical feasibility indicates the best plan is to develop and produce this process internally and is a low-risk scenario as the firm has the resources and experience to succeed. The completed project will stay true to the organization's culture by producing an integrated machine that retains attributes of high quality, safety, efficiency, ease-of-use, and technical reliability and repeatability.

Work Breakdown Schedule

A WBS for the project was created to highlight the major tasks and associated durations and resources necessary in order to accomplish the project. It is separated into two segments to accommodate spacing constraints. The tasks span from the original project concept and approval, through the final implementation of all aspects, including the programming interface application, associated database, and supplemental website. Once the WBS was created and edited, a Gantt chart was created using Microsoft Project.

S.No	Task	Responsible	Duration Totals	Duration (Hrs) / Task	Preceding Tasks
1	**Project Corporate Approval Accepted**	Firm	1	1	
2	**Develop Project Idea**	Tilden May	8	8	1
2.1	*Submit Project Proposal For Approval*	Tilden May	1	1	2
3	**Perform Feasibility Analysis**	Tilden May	8		1
3.1	*Technical Feasibility*	Tilden May		1	1
3.2	*Financial Feasibility*	Tilden May		6	1
3.3	*Organization Feasibility*	Tilden May		1	1
4	**Formulate the Work Plan**	Tilden May	8		3
4.1	*WBS*	Tilden May		4	3
4.2	*Produce Gantt Chart*	Tilden May		4	3
5	**Develop an Analysis Strategy**	Tilden May	8		4
5.1	*Requirements Definition*	Tilden May		8	4
6	**System Architecture / Process Design**	Tilden May	92		3
6.1	*3D CAD Modeling*	Cross Functional Team		80	3
6.2	*Hardware / Software Specification*	Tilden May		8	3
6.3	*Network Diagram*	Tilden May		4	3
7	**Procurement (Quoting / Purchasing / Receiving)**	Tilden May / CFT	320		6.1,6.2
7.1	*Purchase Hardware Items*	Tilden May / CFT		80	6.1,6.2
7.2	*Purchase Software Items*	Tilden May / CFT		40	6.2
7.3	*Tool / Die Machining Components*	Cross Functional Team		80	6.1
7.4	*Fabrication*	Cross Functional Team		120	6.1

8	**Installation of Hardware Components**	Cross Functional Team	120	120	7.3,7.4
9	**Installation / Wiring of Electrical Components**	Cross Functional Team	120	120	7.1,7.4
10	**Program Development**	Tilden May / CFT	144		6.2
10.1	*Develop Windows Application*	Tilden May		80	6.2
10.2	*Develop Machine Side PLC Program*	Cross Functional Team		40	6.2
10.3	*Develop HMI Program*	Cross Functional Team		16	6.2
10.4	*Program Servo Robot Drives (IAI)*	Cross Functional Team		8	9
11	**Configure Cyclone Pro Units**	Tilden May	3		8,9
11.1	*Install Utility Software / Drivers*	Tilden May		1	8,9
11.2	*Install / Configure Cyclone Pro API*	Tilden May		1	8,9
11.3	*Network Node Configuration*	Tilden May		1	8,9
12	**Equipment Integration Debugging / Testing**	Tilden May / CFT	40	40	11.3
13	**Equipment Run-off With Sample Parts**	Tilden May / CFT	8	8	12
14	**Database Development**	Tilden May	40		11
14.1	*Create db / Diagram*	Tilden May		24	
14.2	*Add db connectivity / stored procedures*	Tilden May		8	11
14.3	*Debugging / Testing of data storage*	Tilden May		8	11
15	**Web Development**	Tilden May	24		11
15.1	*Create Web Page for data inquiry*	Tilden May		16	
15.2	*Debugging / Testing of Web Page*	Tilden May		8	11
16	**On site Installation**	Tilden May / CFT	8		
17	**Project Review**	Tilden May / CFT	8		11
			961		

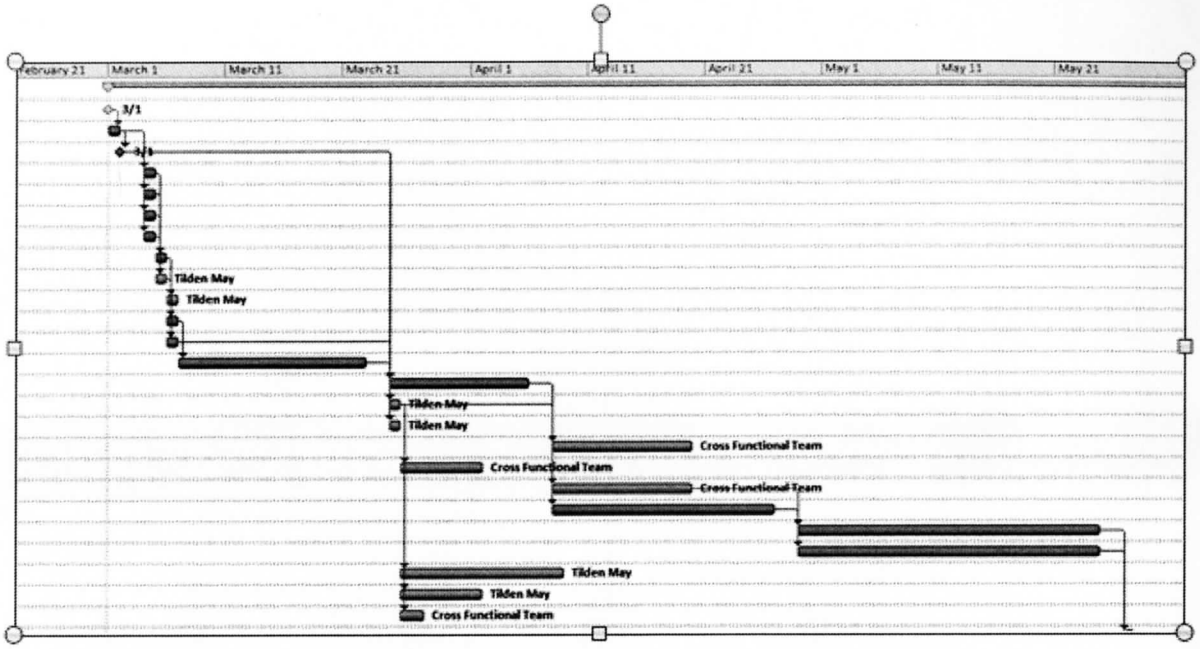
Figure 7: Work Breakdown Schedule

Gantt Chart

Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
Auto Scheduled	**Gantt_Chart**	**77 days**	**Fri 3/1/13**	**Mon 6/17/13**		
Manually Scheduled	Project Corporate Approval Accepted	0 days	Fri 3/1/13	Fri 3/1/13		Tilden May
Manually Scheduled	Develop Project Idea	1 day	Fri 3/1/13	Fri 3/1/13	1	Tilden May
Auto Scheduled	Develop Idea / Submit Project Proposal For Approval	0 days	Fri 3/1/13	Fri 3/1/13	2	Tilden May
Auto Scheduled	Perform Feasibility Analysis	1 day	Mon 3/4/13	Mon 3/4/13	2	Tilden May
Auto Scheduled	Technical Feasibility	1 day	Mon 3/4/13	Mon 3/4/13	2	Tilden May
Auto Scheduled	Financial Feasibility	1 day	Mon 3/4/13	Mon 3/4/13	2	Tilden May
Auto Scheduled	Organization Feasibility	1 day	Mon 3/4/13	Mon 3/4/13	2	Tilden May
Auto Scheduled	Formulate the Work Plan	1 day	Tue 3/5/13	Tue 3/5/13	4,5,6,7	Tilden May
Auto Scheduled	WBS	1 day	Tue 3/5/13	Tue 3/5/13	4,5,6,7	Tilden May
Auto Scheduled	Produce Gantt Chart	1 day	Wed 3/6/13	Wed 3/6/13	9	Tilden May
Auto Scheduled	Develop an Analysis Strategy	1 day	Wed 3/6/13	Wed 3/6/13	8	Tilden May
Auto Scheduled	Requirements Definition	1 day	Wed 3/6/13	Wed 3/6/13	8	Tilden May
Auto Scheduled	System Architecture / Process Design	12 days	Thu 3/7/13	Fri 3/22/13	11,12	Tilden May
Auto Scheduled	3D CAD Modeling	10 days	Mon 3/25/13	Fri 4/5/13	3,12,13	Cross Functional Team
Auto Scheduled	Hardware / Software Specification	1 day	Mon 3/25/13	Mon 3/25/13	3,12,13	Tilden May
Auto Scheduled	Network Diagram	1 day	Mon 3/25/13	Mon 3/25/13	3,12,13	Tilden May
Auto Scheduled	Purchase Hardware Items	10 days	Mon 4/8/13	Fri 4/19/13	15,14	Cross Functional Team
Auto Scheduled	Purchase Software Items	5 days	Tue 3/26/13	Mon 4/1/13	15	Cross Functional Team
Auto Scheduled	Tool / Die Machining Components	10 days	Mon 4/8/13	Fri 4/19/13	14,15	Cross Functional Team

Auto Scheduled	Fabrication	15 days	Mon 4/8/13	Fri 4/26/13	14,15	Cross Functional Team
Auto Scheduled	Installation of Hardware Components	20 days	Mon 4/29/13	Fri 5/24/13	19,20	Cross Functional Team
Auto Scheduled	Installation / Wiring of Electrical Components	20 days	Mon 4/29/13	Fri 5/24/13	19,20	Cross Functional Team
Auto Scheduled	Develop Windows Application	10 days	Tue 3/26/13	Mon 4/8/13	15	Tilden May
Auto Scheduled	Develop Machine Side PLC Program	5 days	Tue 3/26/13	Mon 4/1/13	15	Tilden May
Auto Scheduled	Develop HMI Program	2 days	Tue 3/26/13	Wed 3/27/13	15	Cross Functional Team
Auto Scheduled	Program Servo Robot Drives (IAI)	1 day	Mon 5/27/13	Mon 5/27/13	22	Cross Functional Team
Auto Scheduled	Configure Cyclone Pro Units	1 day	Mon 5/27/13	Mon 5/27/13	21,22	Tilden May
Auto Scheduled	Install Utility Software / Drivers	1 day	Mon 5/27/13	Mon 5/27/13	21,22	Tilden May
Auto Scheduled	Install / Configure Cyclone Pro API	1 day	Mon 5/27/13	Mon 5/27/13	21,22	Tilden May
Auto Scheduled	Network Node Configuration	1 day	Mon 5/27/13	Mon 5/27/13	21,22	Tilden May
Auto Scheduled	Equipment Integration Debugging / Testing	5 days	Tue 5/28/13	Mon 6/3/13	30	Tilden May
Auto Scheduled	Equipment Run-off With Sample Parts	1 day	Tue 6/4/13	Tue 6/4/13	31	Cross Functional Team
Auto Scheduled	Create db / Diagram	3 days	Tue 6/4/13	Thu 6/6/13	31	Tilden May
Auto Scheduled	Add db connectivity / stored procedures	1 day	Fri 6/7/13	Fri 6/7/13	33	Tilden May
Auto Scheduled	Debugging / Testing of data storage	1 day	Mon 6/10/13	Mon 6/10/13	34	Tilden May
Auto Scheduled	Create Web Page for data inquiry	3 days	Tue 6/11/13	Thu 6/13/13	35	Tilden May
Auto Scheduled	Debugging / Testing of Web Page	1 day	Fri 6/14/13	Fri 6/14/13	36	Tilden May
Auto Scheduled	On site Installation	1 day	Wed 6/5/13	Wed 6/5/13	32	Tilden May

Auto Scheduled	Project Review	1 day	Mon 6/17/13	Mon 6/17/13	37	Cross Functional Team



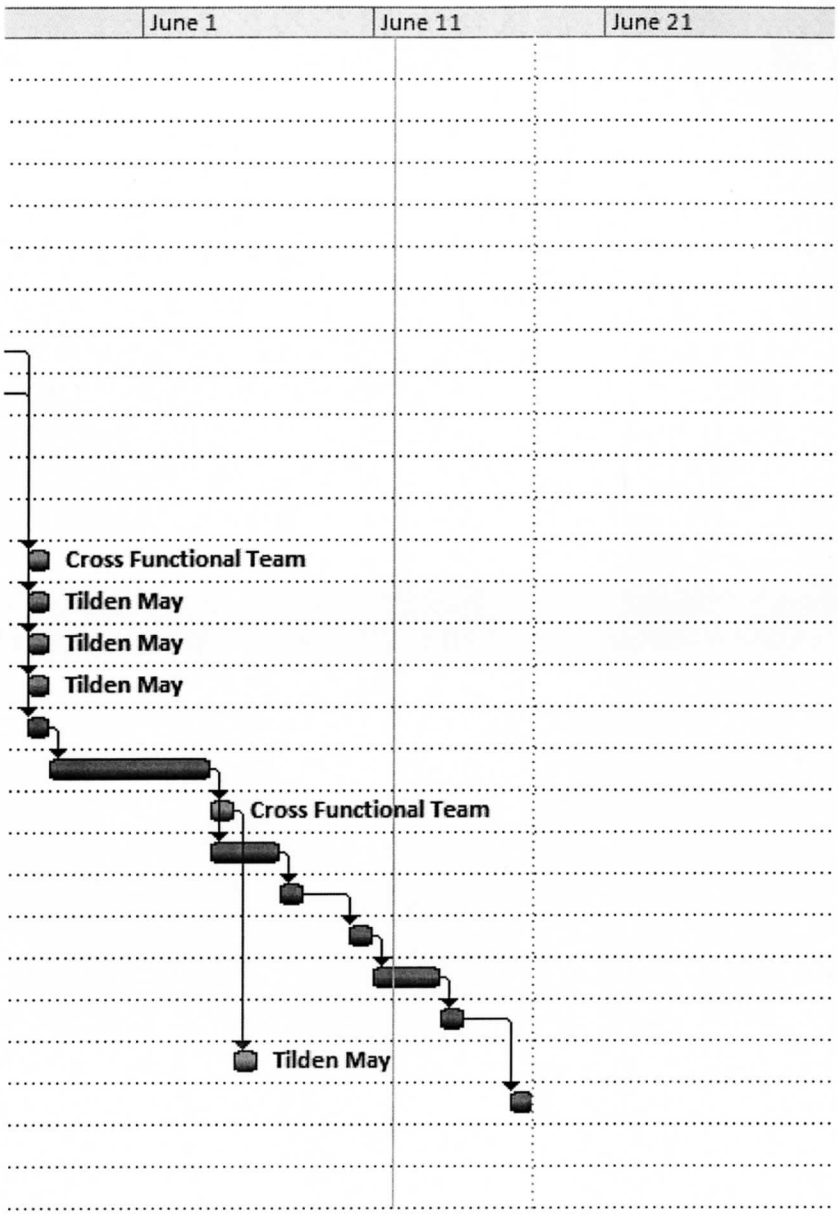


Figure 8: Gantt Chart

Requirements Definition

The refinement of the WBS is an iterative process as tasks are added, adjusted, and expanded as needed throughout the project. However, once the basic steps of the planning phase were determined, the focus turned to establishing the primary requirements for the main system component; the flashing station integration application. The automated flashing station will contain four Cyclone Pro flashing units. These units have built-in memory and can store the configured programming file that is to be uploaded into the onboard microprocessor on the ECU's (product) that are being manufactured. The integration application's main function is to interact with the Cyclone Pro's flashing units. The units use a configured file known as an image which contains the instructions that are to be written onto the microprocessor. The primary interactions include; being able to load specific image files, as selected from any given Windows directory path, executing commands to erase and write the file, and retrieving results upon completion of the flashing cycle to determine whether the attempt was successful or not.

Additionally the application must interact with the base machine controls (HMI) which handles the input and output flow of product in an automated assembly line method. The machine must inform the application when a product is in position and ready to be flashed. The application in return must provide the machine controls with feedback of the results of the flashing cycle so the product can be handled appropriately. Lastly product barcode data must be exchanged in order to track the identity of all flashing transactions and the information shall be recorded in a database. The list below details the system requirements, both functional and nonfunctional for the project.

Functional Requirements

1. Upload Image File

    1.1 The user needs to be able to select a SAP file (compiled programming image file) from the Windows directory.

    1.2 Only valid files should be able to be selected.

    1.3 The user can then upload the image file into the four Cyclone Pro Modules.

1.4 The system should read back the image that is stored in the Cyclone devices and display to the user via the interface.

2. Transact With Cyclone Pro Units

2.1 Open a communication handle with the Cyclone Pro Units via Ethernet.

2.2 Initiate a start programming sequence

2.3 Verify that the units have started (In progress).

2.4 Obtain results upon a completed cycle. (Success or Failure).

2.5 Reset the Cyclone Pro Units if necessary. (Error Condition).

2.6 Close the communication connection upon the application shutting down.

3. Transact With Pro Server

3.1 Inform the HMI (Human Machine Interface) whether the application is active or not. The machine should not function automatically unless the interface application is running.

3.2 Inform the HMI that a programming sequence has begun. This should occur for both lanes 1 and lane 2.

3.3 Inform the HMI of the programming results (Success / Failure). This should occur for all Cyclone Units (1 ~ 4).

3.4 Poll to receive a start request command from the HMI once a pcb is in position and ready to be programmed. This should occur for both lanes (1 & 2).

4. Maintain Product Programming Results

4.1 The system should obtain the product identity via the reading of the unique data matrix barcode residing on each pcb. This ID code along with the programming results should be stored in an SQL database on a Production Traceability Server.

Non Functional Requirements

1. Operational Requirements

   1.1 The program interface application should allow a user to 'change over' the microprocessor programming machine by uploading a Freescale programming file to the Cyclone Pro devices. This should be done in a user-friendly automated method.

   1.2 The application should provide an 'auto-start' option to control product flow and the programming sequence in an automated manner suitable for mass production within an electronics manufacturing environment.

   1.3 The application should include manual controls that provide the user the ability to start and cancel a programming sequence on either lane 1 or lane 2. While the system is running in an automatic mode, the manual controls shall be disabled.

2. Performance Requirements

   2.1 The automated system has an equipment cycle time requirement of 30 seconds per piece. The machine is capable of programming four products at a time so the throughput cycle is 120 seconds / 4 = 30 seconds. Thus the application processing should be optimized in order to aid in achieving this target.

3. Security Requirements

   No special security requirements are anticipated.

4. Cultural and Political Requirements

   No special cultural or political requirements are anticipated.

CHAPTER 4

SYSTEM: DESIGN AND IMPLEMENTATION

System Hardware / Software Specification

Once the core planning phases were in place it was time to begin the design phase. One of the initial tasks that had to be determined was to specify the hardware and software items that were to be used. With there being three main components; a client side integration application, database services, and a web app, it was important to keep in mind the scope and manner that they would serve together as a complete integral system. From an organizational standpoint, the component that was most critical and had to be completed on schedule was the installation and functional operation of the flashing station itself. Thus, the integration application needed to be focused on first before turning attention and man hours to the database and web app services.

The system architecture developed for the DLCM Programmer Interface is comprised of a Client – Server model. The client pc is part of the machine automation and will run the developed custom Windows form application that performs service level integration with the Cyclone programming modules, the HMI touch panel, and additionally handles database transaction requests. The application development language is VB.Net using the .Net Framework 4.0. The database server will perform the data logic and storage role. The server runs on Microsoft Windows Server 2008 Standard and Microsoft SQL Server is the relational database utilized. The server HDD hardware is a RAID 5 drive configuration to protect against data loss and also a daily backup schedule is incorporated. An intranet website will supply user data by providing an interface for querying the database and generating product results. The website will be hosted off the same server as the database and will run via GlassFish server and incorporate JSP and Servlet technology. There are some supplemental components integral to the machine functionality that are provided in the list below; the Cyclone Pro modules which perform the flashing of the microprocessors, the ProFace HMI which contains the machine-side PLC control logic, and the DataMan ID reader which decodes the data matrix barcode residing on each product.

Table 5: Hardware and Software Specifications

	Client Station	Database / Web Server	Cyclone Pro	ProFace HMI	Dataman Barcode Reader
Operating System	• MS Windows 7 Professional	• Windows Server 2008 Standard	N/A	N/A	N/A
Model	• Dell OptiPlex 790	• HP Proliant DL380 G7	• Cyclone Pro V8.47-1.1	• AGP3400	• Cognex DataMan 302
Requisite Software	• MS Visual Studio 2010 Premium • .NET Framework 4.0 • Cyclone Pro Utility / Drivers / API Package • DataMan 4.4 Setup Tool • ProFace GP Pro EX 2.7.0	• Microsoft SQL Server 2008 R2 • SQL Server Management Studio Ver.10.5 • GlassFish Server 3.1.2.2 • JDK 1.6 • JRE SE 7 • JDBC SQL Server Driver	N/A	N/A	N/A
Hardware Specifications	• 120GB HDD • 4 GB RAM • X86 Core i3 CPU • 10/100 Full Duplex Dual NIC	• 2 TB Raid 5 • 12 GB RAM • 64 Bit E5645 6 Core Processor • 10/100/1000 Mbps Full Duplex NIC	• Ethernet / USB/ Serial Comm • 1.8 ~ 5.5 V Target Range • Compact Flash Memory • LCD Screen	• Color Display: 65,536 • 8MB Flash EPROM • 802.3 10/100 Ethernet • Flex Network I/O Modules	• High Resolution (1280 X 1024) • 1D / 2D Barcode Decoding • 24V / I/O/ LED Lighting
Network	• 1000 Mbps Ethernet • VLAN Configured Switch	• 1000 Mbps Ethernet • Backup / Recovery Schedule	• 1000 Mbps Ethernet • VLAN Configured Switch	• 1000 Mbps Ethernet • VLAN Configured Switch	• 1000 Mbps Ethernet • VLAN Configured Switch

Activity Diagram

Focusing on the requirements defined for the flashing station application, an activity diagram was created to capture the structure and primary functions of the system. Upon launching the application, an attempt to connect to each of the Cyclone units via the API (Application Prgoramming Interface) and establish a handle will be made. The handle will then be used for any subsequent calls to the individual units as long as the application is active and a call to disconnect the units has not been made. The arguments passed to establish the handle include passing to the Cyclone the connection method, Ethernet, and the node IP address. Once the connection is complete for all units the current image file stored in the Cyclone memory is retracted and displayed on the user interface. This is done by another function call to the API. At this point, the application is in ready state to perform flashing instructions. If the current image file in use is the EEPROM program that is desired to be used then the user can proceed. However if it is not, the interface uses an OpenFileDialog class (Open File Dialog Class) to allow the user to browse to a directory path on the client and select an appropriate SAP file (Stand Alone Programming) based on the hardware / software configuration of the DLCM product being manufactured at that time. The user has two options to perform flashing instructions, either manual or automatic. Typically the selection will always be automatic for normal mass production operation. The manual is primarily a means to debug, troubleshoot, or individually flash a panel in special circumstances. Functionally the main difference is if manual selection is chosen, the need to interact with the machine side HMI is not necessary and thus no API call's for that are initiated. Since the automatic method is the normal and most used method the process details for that path are explained. Once automatic is chosen, two background worker threads are initiated to poll the HMI using the API ProServer. The first polls for lane 1 programming requests and the other polls for lane 2 request. The main thread of the interface is accessible if needed by the user. Once product flows into the machine and the data matrix code has been captured and the product has been fixtured, the HMI requests to the application that it is ready for a programming sequence to begin. Once this request is captured within the applications worker thread, the thread will come to an end and initiate a new background thread that performs the actual flashing sequence commands to the Cyclone Pro units. The primary steps to instruct the Cyclone Pro to perform flashing are an API call to start the sequence, a call to check status

and confirm the units have indeed started, and then subsequent calls to check the status and determine when the procedure is complete. Once a unit is no longer busy a call is made to obtain the results. Without getting into a deep explanation of the sequence and structure of electrically flashing an EEPROM, the main thing to note is for this particular product, the sequence takes about two minutes to complete. During this time a security check is made, the microprocessor is erased, programmed, and finally verified. The programming is the bulk of the time followed by the verification step. With results in place, two functions will occur, updating the results back to Pro Server using its API and calling functions to perform the database transactions to store the results for the particular DLCM module. Once the HMI receives the outcome of the flashing sequence, it handles the product accordingly. If the DLCM units passed (two products are flashed per lane) the panel of boards will flow downstream out of the machine to the next process for mass production. However, if either unit failed to flash correctly, the machine will divert the panel to a reject station for it to be inspected and evaluated to determine the problem. This sequence happens for both lanes and they operate in parallel with each other so at any given time a sequence can be beginning or ending in lane 1 or lane 2. Once the flashing cycle is complete, if the Cyclone Pro is in an error state, a call to reset the unit is made. Otherwise, if no error state exists, the entire cycle is ready to be repeated over again and the activity returns back to polling Pro Server awaiting the next DLCM product ready to be flashed. Optionally, the user can select to disconnect from the Cyclone Pro units and close down the application. This process and sequence of events is depicted below in the system activity diagram.

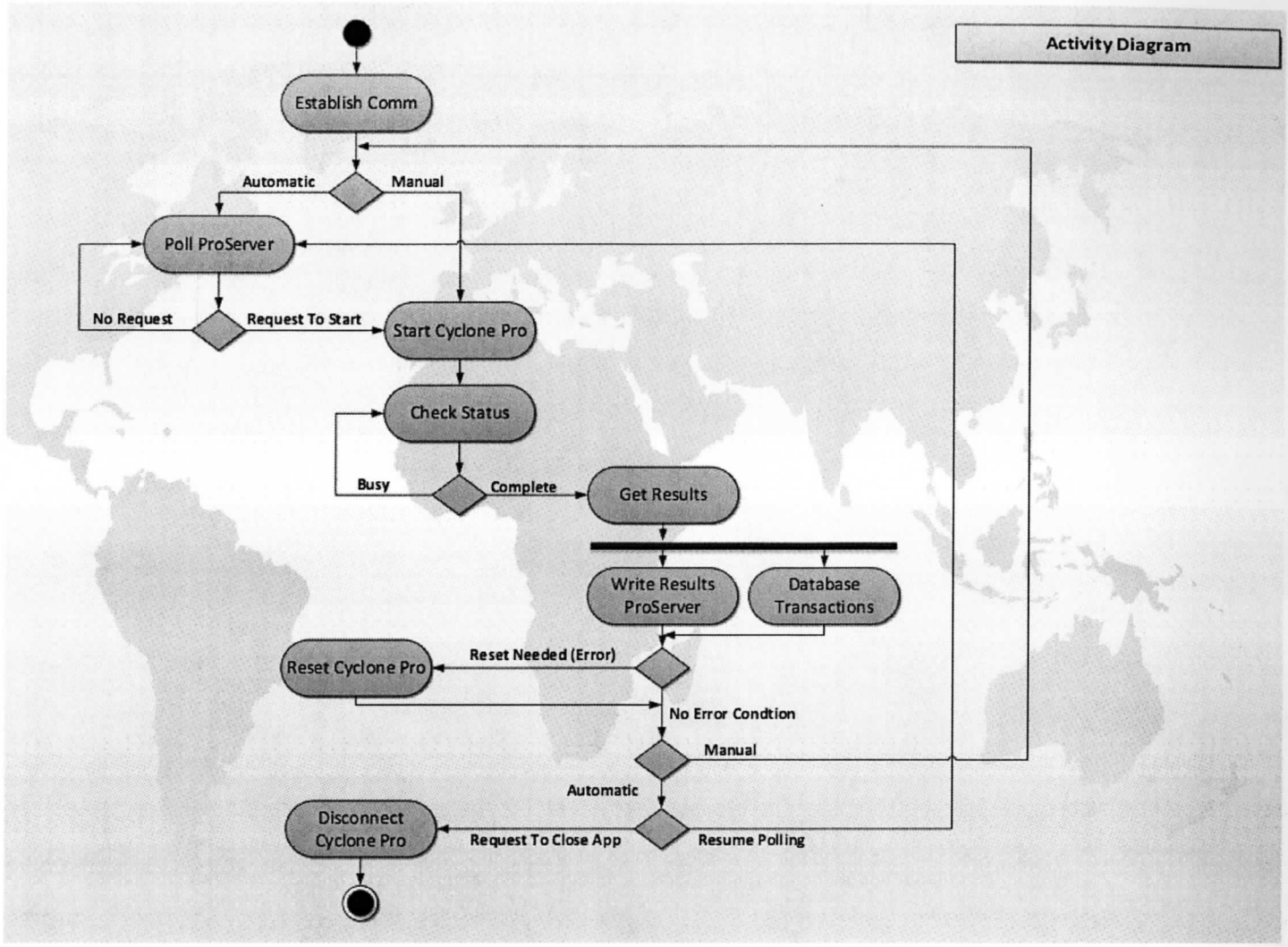


Figure 9: System Activity Diagram

Functional Diagram

In evaluating the primary functions that the system integration application should accommodate, there are two core processes that are the basis for the system framework. The application must interact with the ProServer HMI, which subsequently controls the machine side automation. Secondly, the application must interact with four Cyclone Pro units, which in turn will perform the actual flashing instructions onto the product microprocessor. This is accomplished by using an API / library package that is supplied in both cases by the product vendor. The diagram below depicts this process functionality.

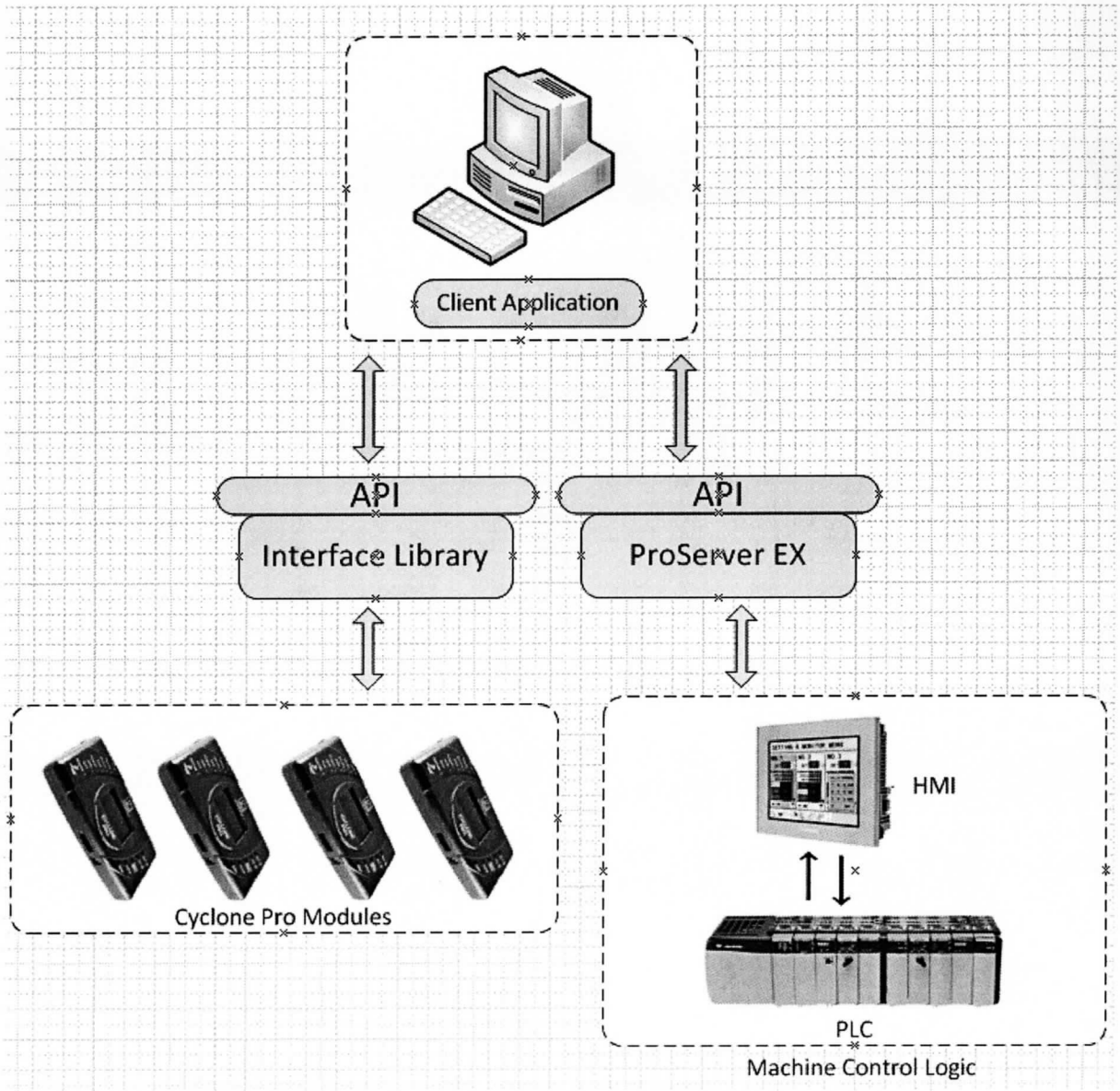


Figure 10: System Functional Diagram

Multithreading

The core functionality of the integration application is to continually poll and interact with the Cyclone Pro flashing units, while at the same time interact with the Pro Face HMI, which handles the machine side controls. There are four total Cyclone Pro units, and they are segmented in sets of two. The machine has two lanes, identified as lane 1 and lane 2. Lane 1 utilizes Cyclone units 1 and 2 while lane 2 uses units 3 and 4. The DLCM product is produced as a panel that contains two circuit boards. So when a panel is supplied into each lane, two circuit boards will be flashed together at the same time. Panels can enter, exit, and be flashed at either lane at any given time. The flow depends on the supply and demand from both upstream and downstream products. With this process flow, the transactions that occur in lane 1 need to operate independently from the actions of lane 2, and vice versa.

The interactions that occur are handled via ethernet using API's for the flashing units and the HMI, and require a state of constant hand-shaking between the peripheral devices and the application. Knowing this and wanting to maintain a responsive graphical user interface for the user, the chosen approach was to use multi-threading to handle the API calls outside of the main interface thread. A total of four background worker threads were implemented to accommodate the services needed. For each lane, a worker thread will handle the transactions with Pro Server with the primary purpose of knowing when to initiate a flashing process for that lane. Additionally for each lane, a worker thread will handle the transactions with the Cyclone Pro modules to perform the flashing sequences and commands that program or flash the microprocessor. This allows the main thread to handle the interface services without instances of locking up or freezing or being unresponsive to the user. The worker thread object instantiates from a .Net built-in class, "BackgroundWorker" which is under the System.ComponentModel namespace. (Background Worker Class) There are three primary events associated with this class. The event 'DoWork' is called to initiate the thread and the actual processing actions take place within this routine. An event 'ProgressChanged', allows periodic GUI updates if needed. While the worker thread is handling its actions, any property updates or data exchanges that need to be sent to the GUI can be executed using this event. Finally at the conclusion of the thread activities, the event 'RunWorkerCompleted' is raised. To prevent potential race conditions where competing threads access a common function or a data element in memory, thread locks were inserted using the SyncLock method

where applicable. (Thread Synchronization (C# and Visual Basic)) A diagram below shows the structure and flow of the processes and the associated threads.

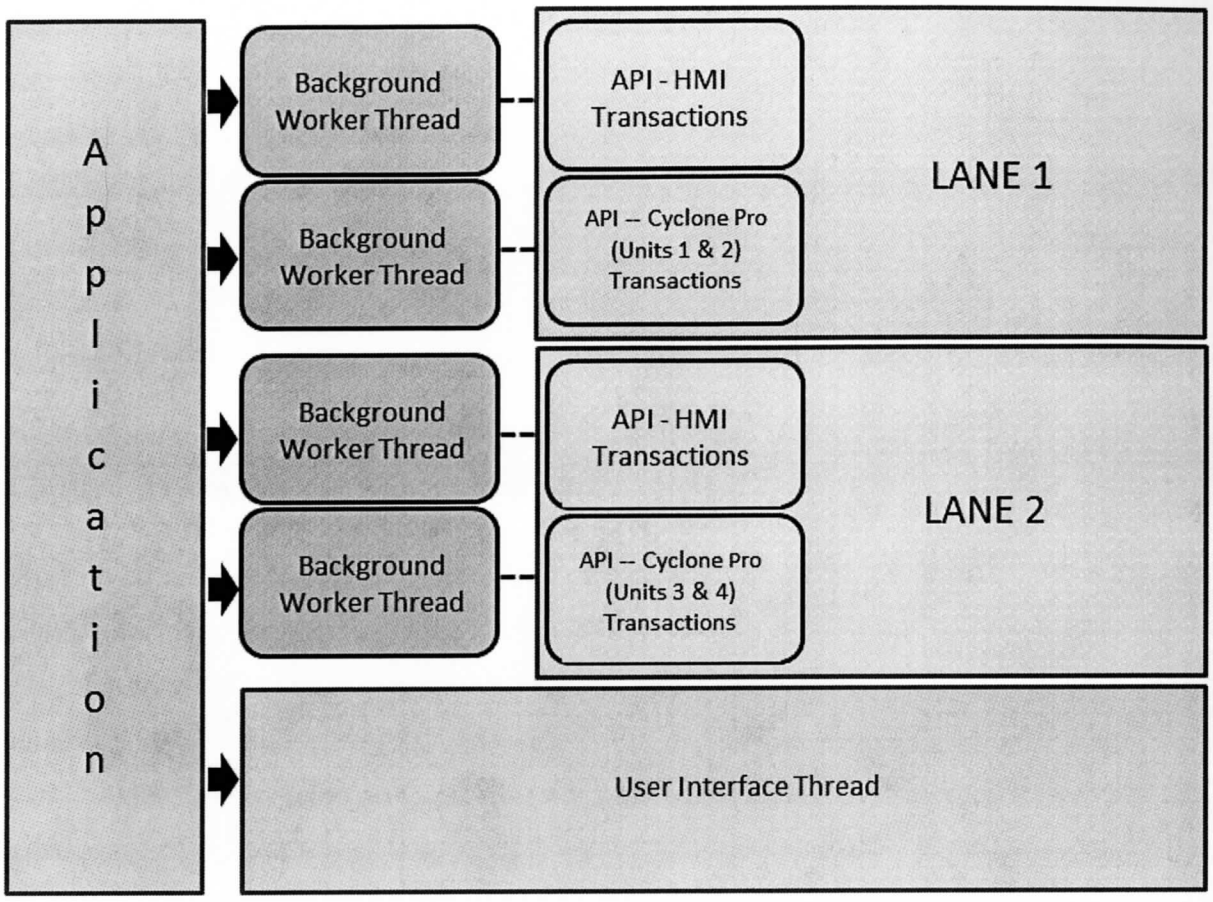


Figure 11: Worker Thread Diagram

User Interface Design

The user interface is a single form with a presentation layout using group boxes, split containers, and panels to provide an easy to use tool. The split container separates lane 1 details from lane 2. Relevant and necessary information is provided to the user regarding the Cyclone unit properties, flashing status, product identity, and the conclusive results. Controls on the form include buttons to launch a file access dialog and buttons and checkboxes to manually and automatically initiate the aforementioned flashing processes. An image of the user interface is shown below.

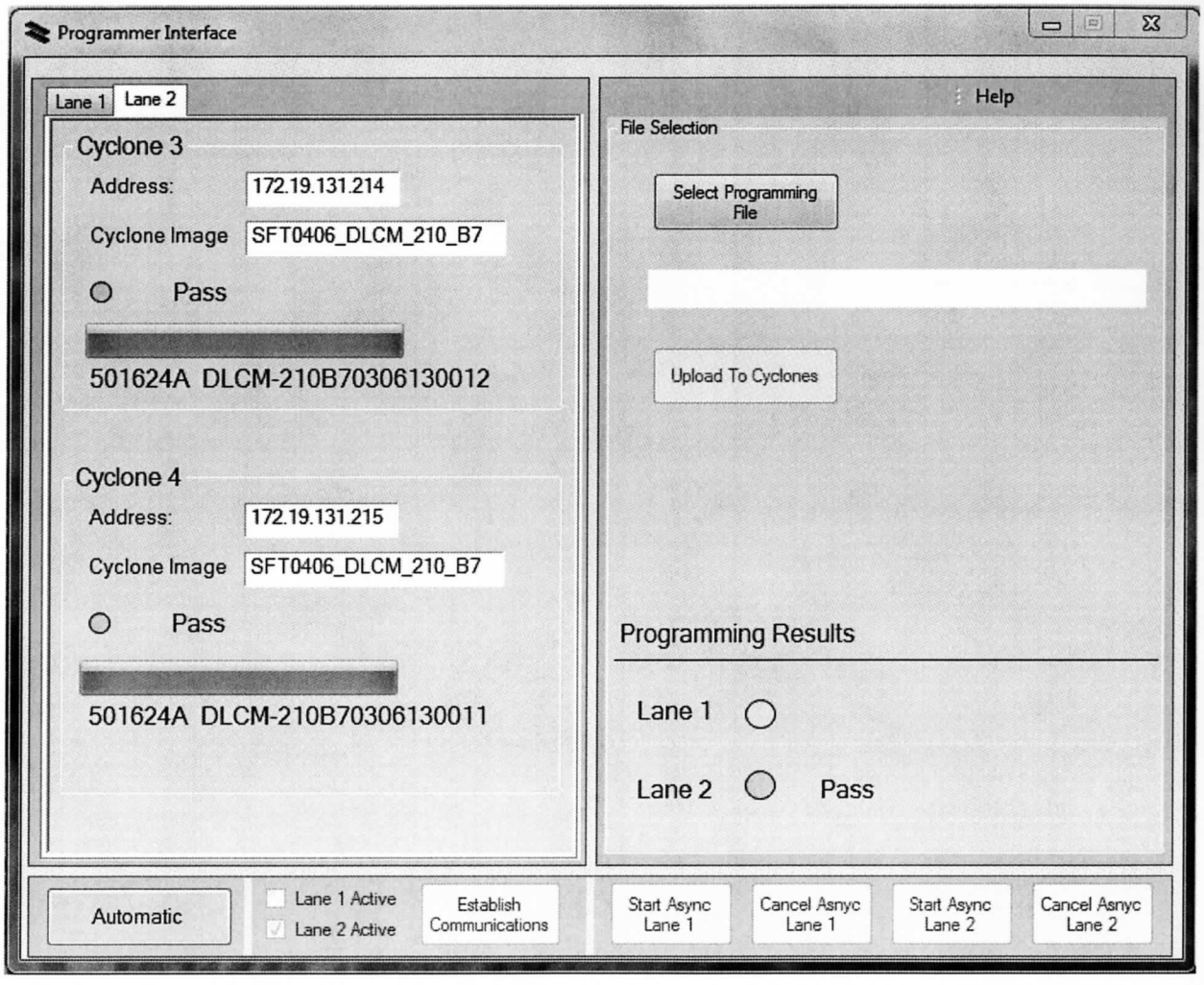


Figure 12: System Graphical User Interface

Database Development

The purpose and need for a database for this project is to have a method of capturing and maintaining the records for each flashing attempt of the product as it goes through this process. This information can be used for quality control checks, to ensure the product correctly received successful EEPROM (EEPROM Definition) flashing and that it was flashed with the correct algorithm. It also can be used as a tool for investigations in the event a part is returned for a field failure as a warranty return. Often circumstances arise that require information to be gathered by production and quality staff, about all of the process steps, results, and time instances for a given product. The inclusion of a database for this automated flashing station will serve and fulfill all of these needs.

The size and scope of this database is quite small. It primarily consists of data to be collected from two main sources; the DLCM circuit board being flashed, and the Cyclone Pro module doing the flashing. The tables created in the database are comprised of attributes stemming from these two sources. For each product produced, a two dimensional barcode is applied by laser as the very first process while the pcb is still a raw unpopulated card. A data matrix code is widely used in industry due to its small size and high rate of reliability. (Data Matrix Code) The barcode content identifies the product by hardware type, software version level, date of production, and finally a serial number. An example of the data matrix code is shown below.

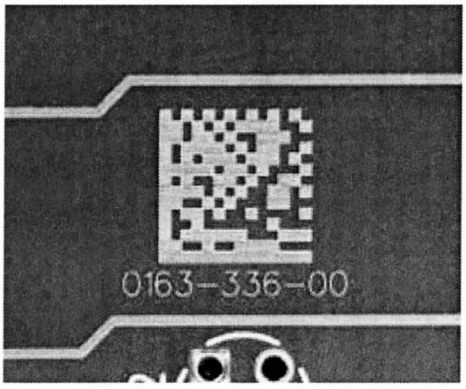


Figure 13: Data matrix Barcode Sample
(FR4 Circuit Board Marking with CO2 Lasers)

The product barcode identity along with attributes of the Cyclone flashing units and the disposition results collectively make up the data to be stored. The initial step was to list all of the data elements needed in an un-normalized table. The comprehensive item list is then separated out into tables to group associated attributes and to eliminate duplicate data entries with in a field. The outcome was four tables as follows; A 'Product_Serial" table that has the attributes of the product identity from the data matrix code, a "Cyclones" table that has attributes about the Cyclone Pro flashing units, a "Image_File" table that contains attributes about the programming file to be flashed, and finally a "Production_Results" table that captures the results information along with foreign keys relations to the other tables. The list below details the tables, keys, and relationships.

Table 6: Database Elements

Table	Fields	Data Type	Key	Description
Cyclones	unit_ID	tinyInt	PK	Identifies the Cyclone Pro Unit (1~4)
	Lane	tinyInt		The machine lane in which the product was processed
	ip_address	varChar(15)		The static address of the Cyclone Unit
Image_File	image_ID	Integer	PK	ID number associated with the cyclone image file name
	cyclone_image	varChar(120)		The image file name that is flashed onto the microprocessor
Product_Serial	serial_ID	Integer	PK	ID number associated with the datamatrix barcode
	datamatrix_barcode	varChar(40)		The unique product identity of the DLCM circuit board
Production_Results	result_ID	Integer	PK, FK	Composite PK with date_time; FK to Product_Serial (serial_ID)
	date_time	dateTime2	PK	The date and time of the flashing occurrence
	unit	tinyInt	FK	FK to Cyclones Table (unit_ID)
	result	varChar(1)		The conclusion of whether the flashing was successful or not
	cyclone_image	Integer	FK	FK to Image_File Table (image_ID)

The relationship between the tables is shown below in the ERD.

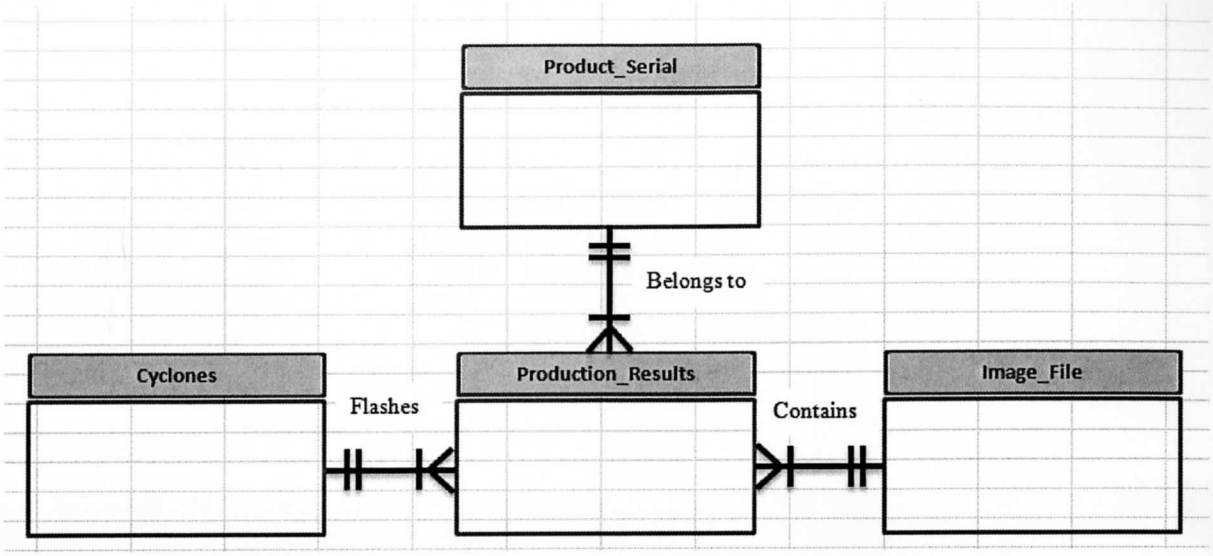


Figure 14: Entity Relationship Diagram

Once the structure of the database was determined, scripts were coded to create the tables and apply any needed constraints. The integration application will update the database at the conclusion of every product being flashed in the machine. Although it is typical for a product (DLCM) to only be processed once through this machine, it is possible for it to be ran and flashed multiple times. This can happen if the product fails to properly flash the first time, or if the machine is being debugged and product flows through multiple times, or if there is a need to re-flash to a different image file. Given this, the "Production_Results" table will get updated each and every time, however the other tables get updated contingent upon whether a record of the product barcode or the specific image file already exists in the database or not. The data describing the Cyclone units themselves is static. There are four flashing units in the machine and the configuration of those units does not change dynamically. Stored procedures were created to do the following; query to see if a product ID (data matrix code) was present in the database and return a response, query to see if an image file existed in the database and return a response, and finally a procedure to update the database. The update procedure uses the returned responses from the first two queries to determine if all tables need updated or only the "Production_Results" table. It then processes the updates accordingly. Before the client application was modified to call the procedures,

test scripts were created to execute each and simulations were ran to check for errors and data anomalies. The project scripts and stored procedures used are shown below.

Table 7: SQL Scripts / Procedures

SQL Script Name	Description
FA5_Create_Tables.sql	Creates all the FA5 tables and constraints
FA5_Insert_Cyclones.sql	A script for inserting 'static' data into this table
FA5_Create_barcodeCheck_Procedure.sql	Procedure for checking to see if a data matrix barcode identity exists in the db
FA5_Create_ImageCheck_Procedure.sql	Procedure for checking to see if a Cyclone Image File exists in the db
FA5_Create_Update_db_Procedure.sql	Procedure for updating all the tables in the db upon conclusion of flashing sequence
Execute_barcodeCheck.sql	A script for testing the procedure 'barcodeCheck'
Execute_ImageCheck.sql	A script for testing the procedure 'ImageCheck'
Execute_Update_db.sql	A script for testing the procedure 'Update_db'

As noted in the hardware / software specifications, the RDBMS used is Microsoft SQL Server 2008. The image below shows the database diagram created under the database FA5, named after the production line moniker.

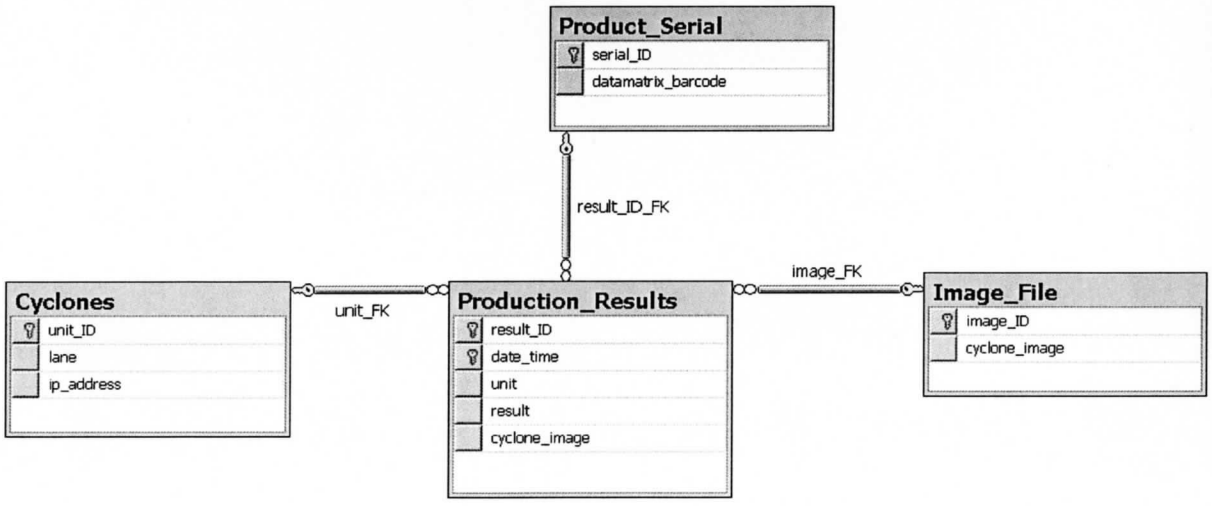


Figure 15: System Database Diagram

Network Infrastructure

The networking infrastructure backbone and hardware items needed for this project were mostly already in existence. Additional patch cable drops were ran during the FA5 line installation phase. A 48 port layer three switch is used for the hosts on FA5 line and a VLAN configuration was put forth for segmenting the nodes on the LAN for routing. Static IP addresses were established for the client pc, four Cyclone Pro units, the Cognex Dataman barcode scanner, the ProFace HMI, and the system server. The server for this system performs the role of data storage and additionally hosts the website that provides users with the system's processed results. This server and the FA5 line (DLCM programmer interface client) are located in separate facilities on the corporate campus. A fiber optic trunk line provides connection between the two. The figure below details the network topology for the system.

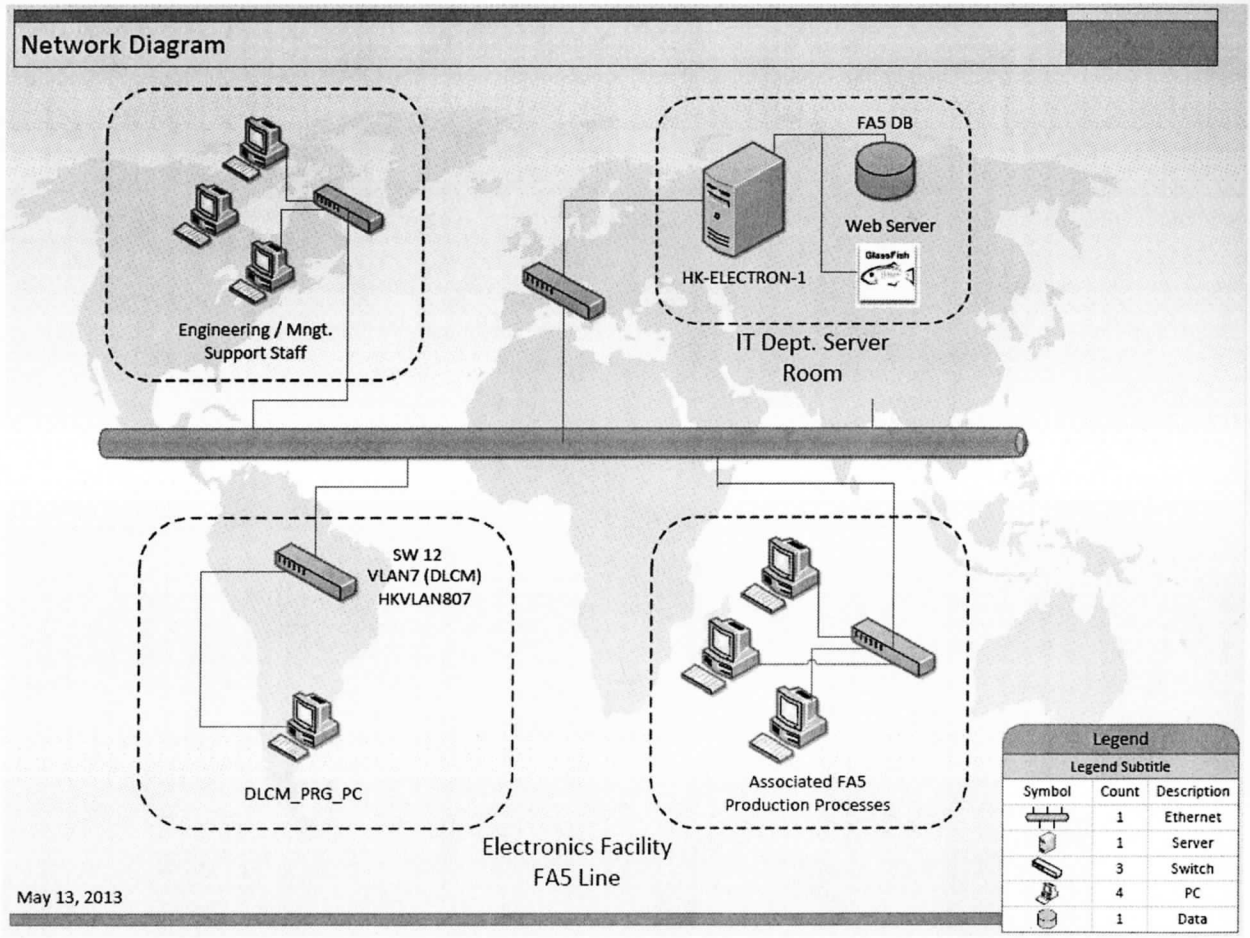


Figure 16: System Network Diagram

Web Development

The purpose for developing the website was to produce an intranet available website that would provide production and quality support staff with quick and easy access to view the flashing results for a given product. The layout approach is straightforward; there is an introductory page that provides an input box for capturing the data matrix barcode of a DLCM unit. The normal method is for a handheld USB scanner to be used to decode the barcode and then supply the string, although the information could be manually entered as well. A submit button would then signify the request to process the results and display on a results web page.

Though the interface and layout of the website is simplistic, the main functionality is the establishment of the SQL connection and processing the query. In determining which technology and language to use, the focus centered primarily on the design to be a server side scripting site as opposed to client side. Fundamentally the site should be dynamically generated, even though it contains only two web pages and the change in content revolves strictly around the information from the database. PHP was considered but ultimately JSP was chosen in order to build upon foundational JAVA programming experienced in INFS 734 (Client / Server Technologies) and to work with NetBeans IDE. Java SDK, including both the JRE and JDK, along with GlassFish can be downloaded from Oracle's website. (Oracle Downloads) Initial steps included coding and exercising the website to work in a test environment, building the project and creating a WAR file. Once the site was operational in a test environment GlassFish application server was then configured onto the system server to be used for hosting the website. The table below highlights the core technologies and software components used during this phase of the project.

Table 8: System Web Technologies

Web Technologies	
IDE	Netbeans 7.3
Java Runtime Environment	JRE SE 7
Java Development Kit	JDK 1.6
SQL Driver	Microsoft JDBC SQL Driver 4.0
Application Server	GlassFish 3.1.2.2

Before full deployment onto the server, the website was created and tested within the Netbeans development environment. The base layout of the main page was created within a JSP file. Inside the project, the libraries were expanded to include the JAR (Java Archive) folders for the JDBC SQL driver (Microsoft JDBC Driver 4.0 for SQL Server) and also the GlassFish application server. Once configured, a servlets java class page was added that would handle the SQL connection and provide the database query results in a displayed table. The welcome page is shown below. The user enters the product id (data matrix code) into the input box on the page. The best method to do this is by using a hand held barcode reader. These are abundant and widely used in our organization. Alternately they can manually type in the information. Once the string is in place, the user selects the 'submit' button.

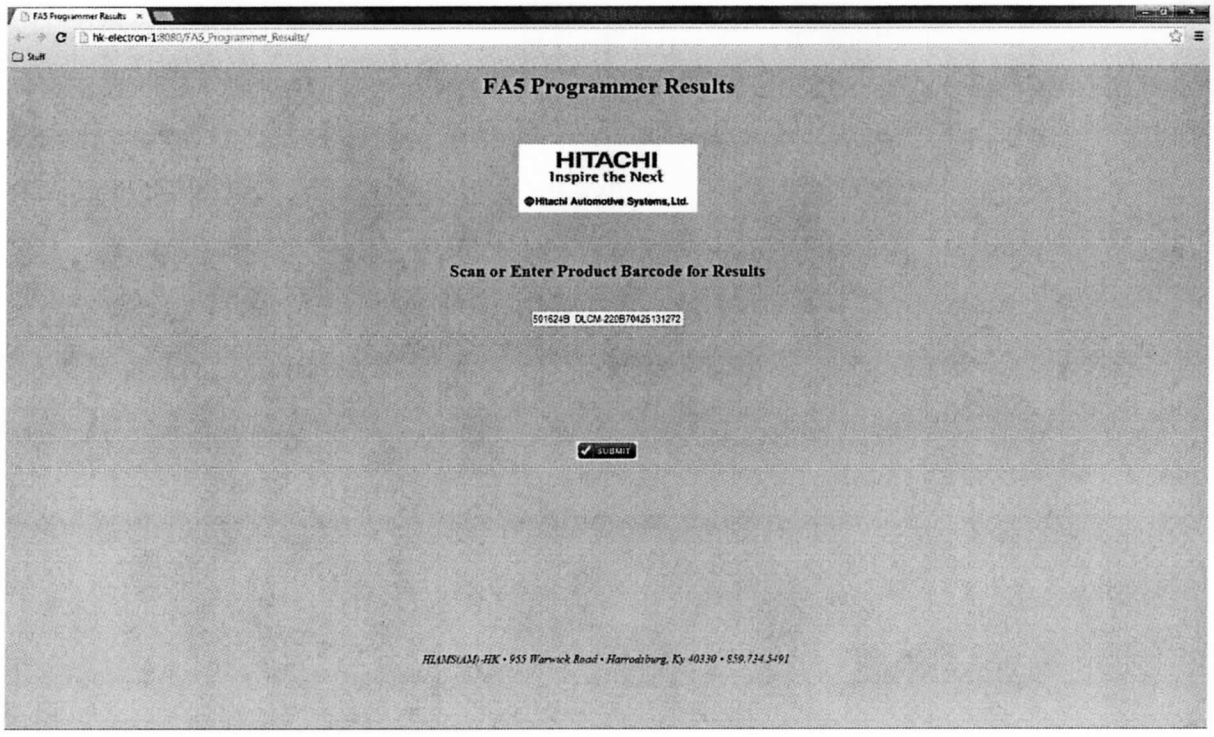


Figure 17: System Website Main Page

The action from the submit button invokes the results servlet. This is where the SQL connection occurs and the string acquired from the user entry is the basis of the query. The results, if any, are displayed to the user in a table. A loop cycle iteration adds rows of information for each record returned from the query. This is done using HTML tags. Some

snippets of code are shown below including establishing the SQL query, instantiating the SQL driver, and populating the table with results.

```
//**********SQL Server Query for providing results snapshot to user*****************************
        String test_query = "Select date_time, result, Image_File.cyclone_image, lane, unit_ID\n" +
"from Production_Results, Image_File, Cyclones\n" +
"where result_ID = (Select serial_ID from Product_Serial where datamatrix_barcode ='" + "'" + getbarcode + "')\n" +
"And Production_Results.cyclone_image = image_ID\n" +
"And unit = unit_ID\n" +
"Order By date_time";
//*********************************************************************************************

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();
String connectionUrl = "jdbc:sqlserver://172.19.132.1:1433;" +
"databaseName=FA5;user=VS1;password=HondaVS1;";
Connection con = DriverManager.getConnection(connectionUrl);

con.getMetaData();
Statement st = con.createStatement();
ResultSet myResult = st.executeQuery(use_query); //execute the applicable query


//loop to populate the table with the query results

 while (myResult.next()) {

     //Add data rows to table for each iteration of the loop; results from SQL query
     out.println("<TR>");
     out.println("<TD><CENTER>");
     out.println(myResult.getString(1));
     out.println("</CENTER></TD>");

     out.println("<TD><CENTER>");
     out.println(myResult.getString(2));
     out.println("</CENTER></TD>");

     out.println("<TD><CENTER>");
     out.println(myResult.getString(3));
     out.println("</CENTER></TD>");

     out.println("<TD><CENTER>");
     out.println(myResult.getString(4));
     out.println("</CENTER></TD>");

     out.println("<TD><CENTER>");
     out.println(myResult.getString(5));
     out.println("</CENTER></TD>");
     out.println("</TR>");


 }
 //close db connection streams
myResult.close();
 st.close();
con.close();
```

Figure 18: Servlet Code Snippets

The resulting response web page is rendered. Based on the identity of the product, the user is informed of; the image file flashed on the microprocessor, the date and time it was processed, whether the flashing was successful or not, which lane of the machine handling occurred, and the unit number of the Cyclone module that performed the flashing.

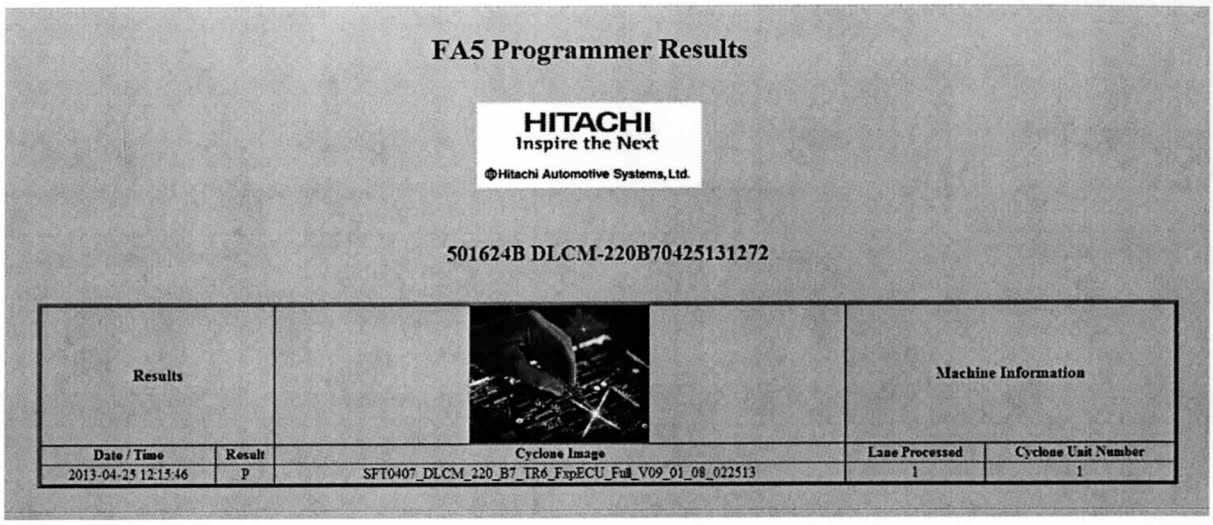


Figure 19: System Website Results Page

Once the site was functional within the Netbeans IDE it was time to configure the project to run on the system server. This involved packaging the project into a WAR file (Web Application Archive), distributing the JDBC SQL driver, and configuring GlassFish server to deploy the WAR when the URL is accessed. A WAR file is used to distribute a collection of files (JSP, Servlets, Java classes, HTML, and others) that collectively make up a web application. (War File Format; Sun Microsystems) The first step was to install and setup the JDBC driver onto the system server. (Deploying The JDBC Driver) The next task was to install GlassFish onto the system server. For GlassFish to work properly with the Java version in use, the domain config file and 'Java_Home' environment variable had to be edited. (Matthias, 2012) Once installed, the domain service had to be started; this can be done from the command prompt by navigating to the directory path for asadmin.bat. (How to deploy web app using asadmin command on GlassFish Enterprise Server, 2009) In this instance, it was installed in "C:\Program Files\glassfish-3.1.2.2\bin\". Run asadmin and initiate the service by the command "asadmin start-domain domain1". The name "domain1"

is a default name and can be changed if desired. These console instructions are shown below and you can see that the service started successfully.

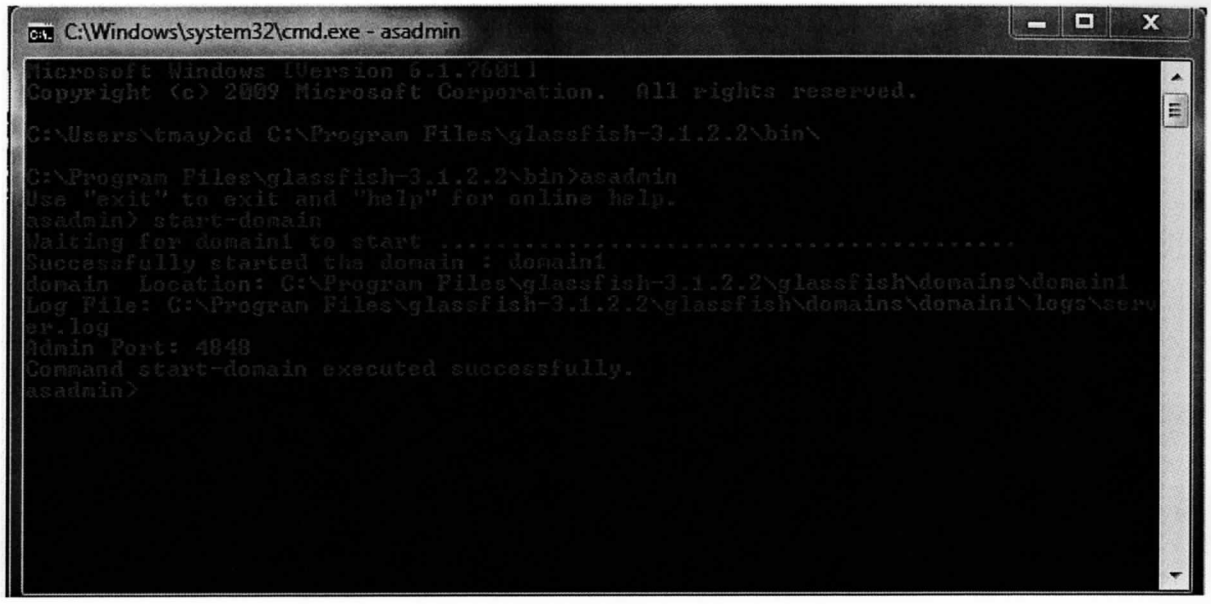


Figure 20: GlassFish Console Command To Start Domain

After the GlassFish server domain has been started you can also verify that the server is up and running via the web browser by navigating to the following URL "http://localhost:8080/". The screenshot below displays this.

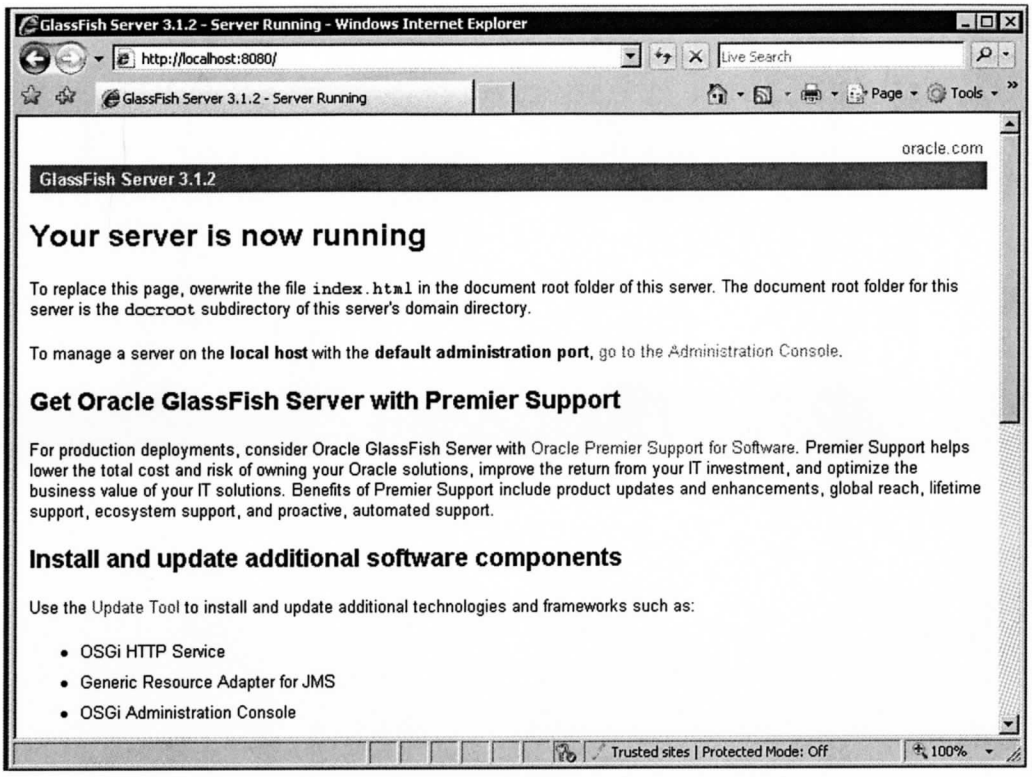


Figure 21: GlassFish 3.1.2 Server Running

Now that GlassFish is installed and a server instance is running, the next thing to do is configure the application server to deploy the website WAR package each time a call to the URL is made. This can be done by again launching a command prompt console and running the asadmin.bat file. There are subsequent commands that can assign and deploy the WAR file. However there is another way to do this and it is more intuitive. Using the URL "http://localhost:4848/", launch the GlassFish server console. A user friendly interface is provided and the application server can easily be configured. To deploy the website, select the Application tab and direct the file path to the location of the website WAR file. Check the deploy checkbox and save the configuration. An example of this is shown below.

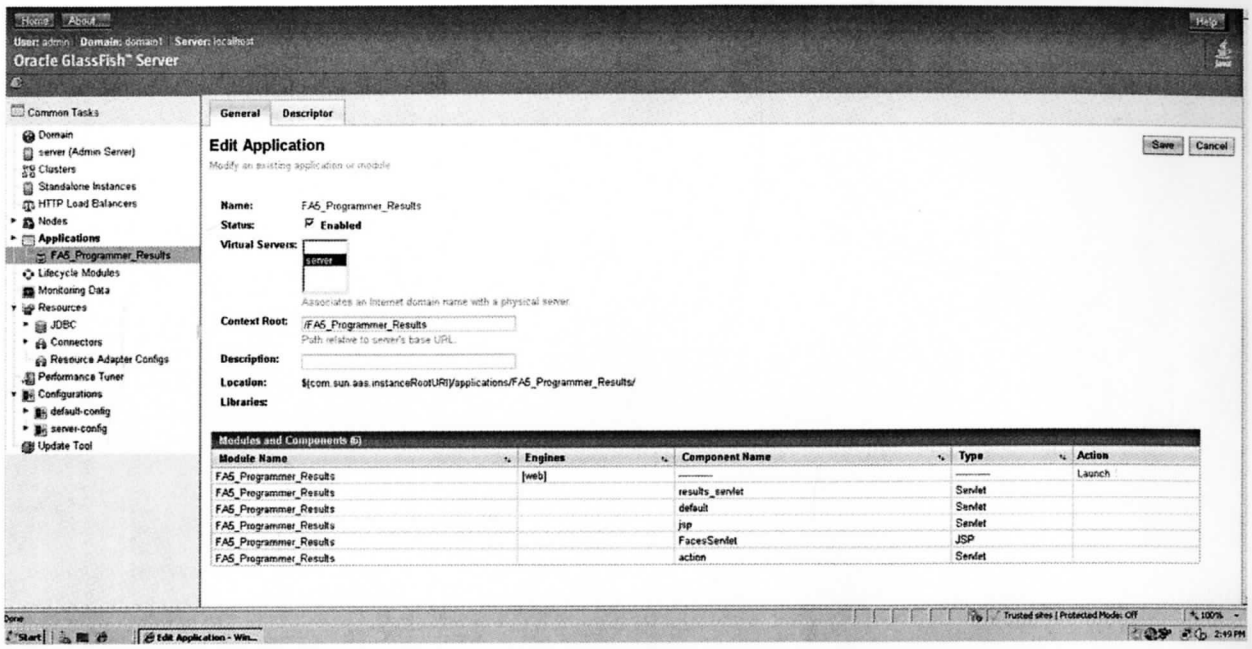


Figure 22: GlassFish Server Graphical Console

After the website has been configured to deploy in Glassfish and the server domain is running, the website can be accessed by clients to the server by navigating to the URL "http://hk-electron-1:8080/FA5_Programmer_Results/". The path includes the server name, port used, and name of the deployed website.

Machine Concept

The driving problem scope of the project was to develop a system that would provide an automated inline method to flash the microprocessors on the DLCM product and do so in a manner suitable for high speed manufacturing. The aforementioned systems all contribute to this case however there was still the need to design a robotic machine capable of handling all of the necessary functions of performing the flashing process. For this portion of the system the firm's internal equipment design group who specializes in machine automation was utilized. The machine was designed to input and output product automatically, flow in a left to right manner, include barcode scanning capability, contain fixtures for electrical connectivity for flashing, be dual lane capable, and support a minimum of four Cyclone Pro units. The three dimensional model below depicts the concept showing the base framework of the machine.

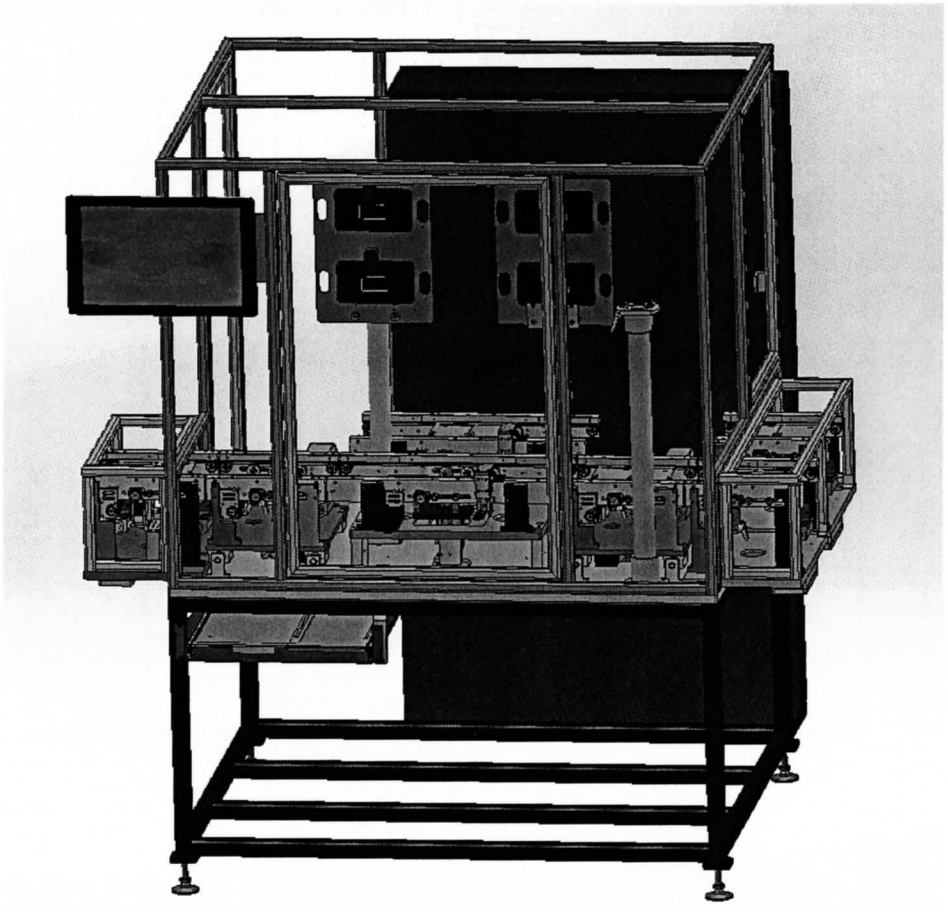


Figure 23: 3D Model of Machine

CHAPTER 5

CONCLUSIONS

As proposed earlier there were three core system objectives for this project. The first was to develop a Windows based application that would integrate with a custom machine in order to automate the process of flashing onboard microprocessors for the DLCM production line. This objective was met and equally as important the deadline for installation and process availability for the production line was achieved. Synergetic team trust was established during the planning and design phase with the cross functional group that was designing and building the machine. Before delivery and installation a week was spent running trials and test runs while debugging the application, machine side controls, and making any enhancements or modifications needed. This piece of equipment along with the entire mass production line was installed on time and has produced customer parts as part of the start-up builds for validation purposes. It will soon launch into mass production producing parts daily for many years to come for Chrysler vehicles.

The second core objective of this system was to develop a database in which the client application would update based on the process transactions for all products being produced in this machine. Once the machine was operational and installed for use, focus shifted towards this objective. Once the key sets of information that needed to be collected were identified, the relationship structure of the database was designed. Scripts were coded to create the database, apply constraints, and form the stored procedures that are used by the application to transact with the database. During this phase tasks to assure the network infrastructure was completed and configured correctly took place. The system server is located in a separate facility than the client pc. Finally updates to the Windows application to perform the database transactions transpired and tests were performed to ensure the system was properly updating the database.

With the database in place and being updated the final objective to be completed was the development and deployment of a system website that our internal support staff could use to access the database and get product specific results. After some contemplation a decision

was made to create the website in the Netbeans IDE and using JSP and servlets. The site was tested within the development environment and once ready, it was deployed onto the server using GlassFish application server. A user can launch the URL, scan or enter the barcode identity of a DLCM unit, and have the details of the flashing process provided to them. Once this was implemented, all three primary phases of main system objectives were complete.

In addition to these core objectives, all proposed deliverables have been met and included in this project report. These include the planning phase: feasibility analysis, work breakdown schedule, and Gantt chart, and the analysis phase: requirements definition; and the design phase: activity diagram, database diagram, network diagram, and a hardware / software specification. Additionally the system source code is included for the client Windows application, the system website, and the database SQL scripts. Lastly a system user manual was created and included that instructs the user on the proper steps and sequence of using the software.

The original brainstorming sessions to conceive this capstone project focused on the desire to do a project that spanned across as many of the MSIS courses as possible. This concept remained prevalent throughout the scope of the project and reflecting on it now once it has been implemented, it is deemed to be successful in that area. With this being a system development project, various software elements were utilized including VB.Net, Java, and HTML. Courses that helped hone these skills include INFS-730 (Programming for E-Commerce), INFS-605 (I.S. Programming), and INFS-734 (Client-Server Technologies). This project also included the modeling and structuring of a system database as well as SQL scripting. Course that helped derive experiences from for this portion include INFS-760 (Enterprise Modeling & Data Management), and INFS-762 (Data Warehouse / Data Mining). This system involved elements of network infrastructuring and LAN and node configurations in which mirrored experiences gained from INFS-750 (IT Infrastructure, & Network Management). Beyond the key granular objectives of this system that was underway, there was also the all encompassing DLCM new production line investment occurring in parallel with this system deadline. Key aspects drew upon for project management along with the fundamental SDLC stages for planning, designing, and implementing the project were the courses; INFS-720 (System Analysis & Design), INFS-724 (Project and Change Management), and INFS-780 (I.T. Strategy and Policy).

The development and implementation of this system proved to be a rich and rewarding project. This opportunity helped to bring together and apply many of the concepts, theories, and skills acquired throughout the graduate program in MSIS at Dakota State University. Technology and the tools society use will continue to evolve at rapid rates. Beyond many of the technical skills students acquire from this program, there are some core business concepts that will provide long lasting benefits during one's career. One is the ideal of being both efficient and effective in the systems that are put in place. To be effective, a system has to accomplish its goals and objectives as they have been defined. Another key concept is establishing, as a firm, what your target business model is and defining what it is that gives you a competitive advantage. This should continually be evaluated and analyzed as strategic company decisions are made and projects and systems are put forth. Finally graduates should recall that there are a high percentage of IT projects that never materialize to meet the objectives or reach an implementation phase. This of course results in a high level of financial loss and is often attributed to poor planning and lack of strategic focus at the very onset of the project.

Graduates should hope and aspire to utilize the skills, knowledge, and experience they have gained through this program to lead and manage others, develop and implement effective and successful technological systems, and promote an environment of continual exploration, growth, and fulfillment in the realm of business information systems.

REFERENCES

Application Prgoramming Interface. (n.d.). Retrieved April 2013, from Wikipedia: http://en.wikipedia.org/wiki/Application_programming_interface

Background Worker Class. (n.d.). Retrieved February 2013, from MSDN: http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1

Chaikin, D. (2004, December 14). *How It Works: The Drive Train*. Retrieved February 2013, from Popular Mechancis: http://www.popularmechanics.com/cars/how-to/repair-questions/1302716

Cyclone Pro. (2013). Retrieved May 2013, from Freescale.com: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=M68CYCLONEPROE

Data Matrix Code. (2013). Retrieved July 2013, from Data Matrix Code: http://www.datamatrixcode.net/

Definition of HMI. (n.d.). Retrieved July 2013, from PCMag: http://www.pcmag.com/encyclopedia/term/44300/hmi

Deploying The JDBC Driver. (n.d.). Retrieved April 2013, from MSDN: http://msdn.microsoft.com/en-us/library/aa342329(v=sql.105).aspx

EEPROM Definition. (n.d.). Retrieved February 2013, from PCMag: http://www.pcmag.com/encyclopedia/term/42403/eeprom

FR4 Circuit Board Marking with CO2 Lasers. (n.d.). Retrieved June 2013, from Coherent.Com: https://www.coherent.com/Applications/index.cfm?fuseaction=forms.AppLevel2&AppLevel2ID=104

How to deploy web app using asadmin command on GlassFish Enterprise Server. (2009, April 13). Retrieved May 2013, from FormDev: http://www.fromdev.com/2009/04/command-line-deployment-at-linuxunix.html

Matthias, H. (2012, September 10). *How to modify the version of Java used for GlassFish 3.1.2.2*. Retrieved May 2013, from My Personal IT BLog: https://matthiashoys.wordpress.com/tag/as_java/

Meisel, J. (2007, December). *Multithreaded Programming*. Retrieved July 2013, from EvaluationEngineering.com: http://www.evaluationengineering.com/articles/200712/multithreaded-programming.php

Microsoft JDBC Driver 4.0 for SQL Server. (n.d.). Retrieved April 2013, from Microsoft.Com Download Center: http://www.microsoft.com/en-us/download/details.aspx?id=11774

Morrison, V. (2005, August). *What Every Dev Must Know About Multithreaded Apps*. Retrieved July 2013, from MSDN Magazine: http://msdn.microsoft.com/en-us/magazine/cc163744.aspx

Open File Dialog Class. (n.d.). Retrieved March 2013, from MSDN: http://msdn.microsoft.com/en-us/library/system.windows.forms.openfiledialog.aspx

Oracle Downloads. (n.d.). Retrieved March 2013, from Oracle.Com: http://www.oracle.com/technetwork/java/javaee/downloads/index.html

Thread Synchronization (C# and Visual Basic). (n.d.). Retrieved April 2013, from MSDN: http://msdn.microsoft.com/en-us/library/ms173179(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2

War File Format; Sun Microsystems. (n.d.). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/WAR_file_format_(Sun)

Woodring, M., & Cohen, A. (1997, December). *Win32 Multithreaded Programming*. Retrieved July 2013, from O'reilley Online Catalog: http://oreilly.com/catalog/multithread/excerpt/ch01.html

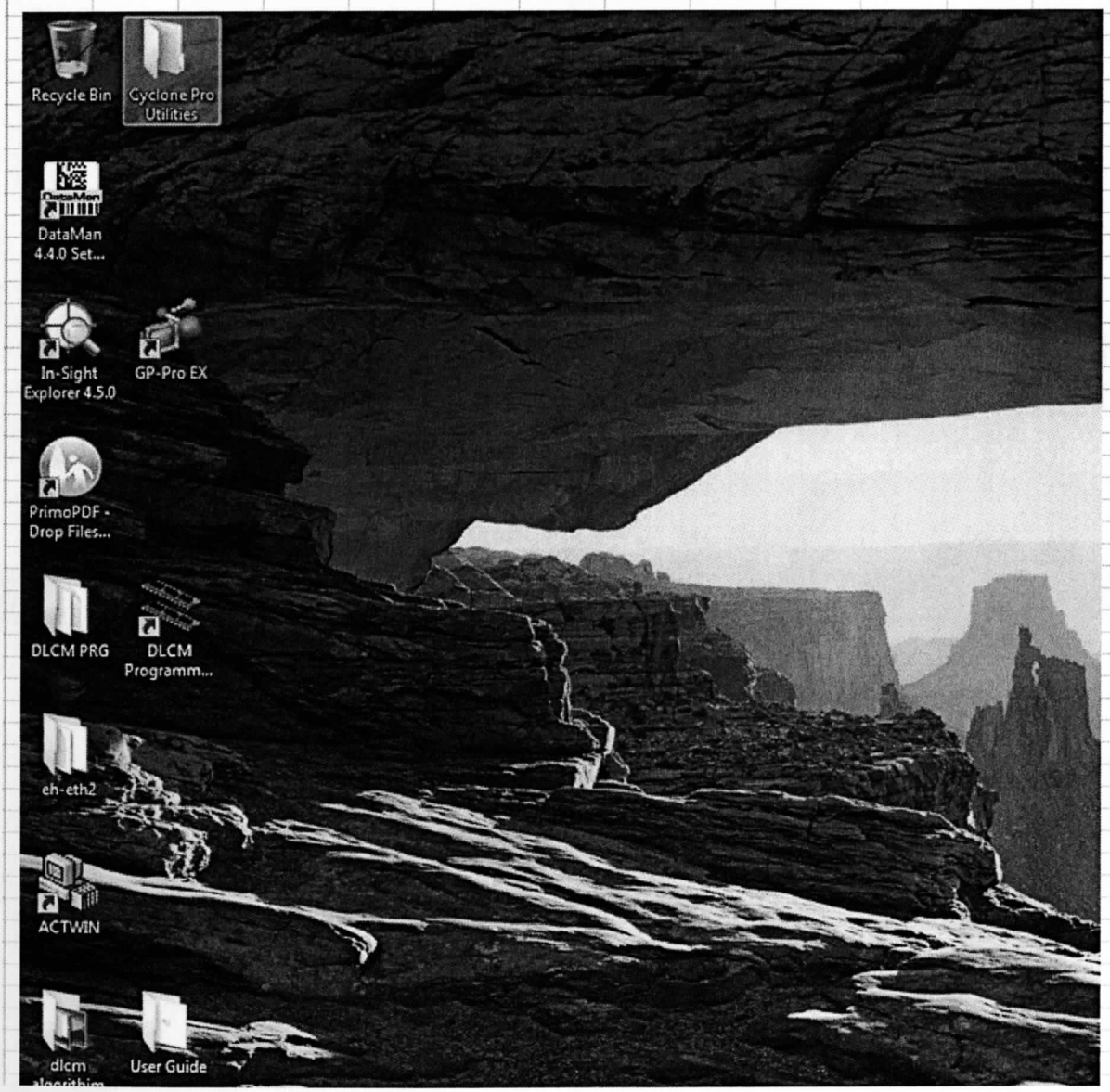
APPENDICES

APPENDIX A: USERS' MANUAL

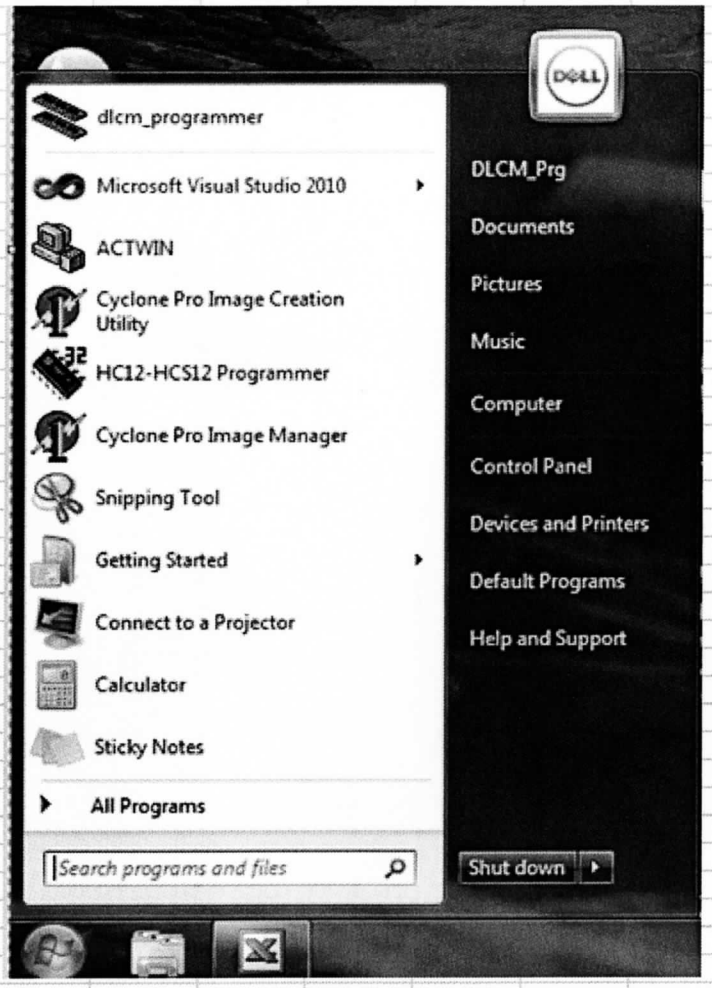
Startup Instructions

I: How To Launch DLCM Programmer Interface

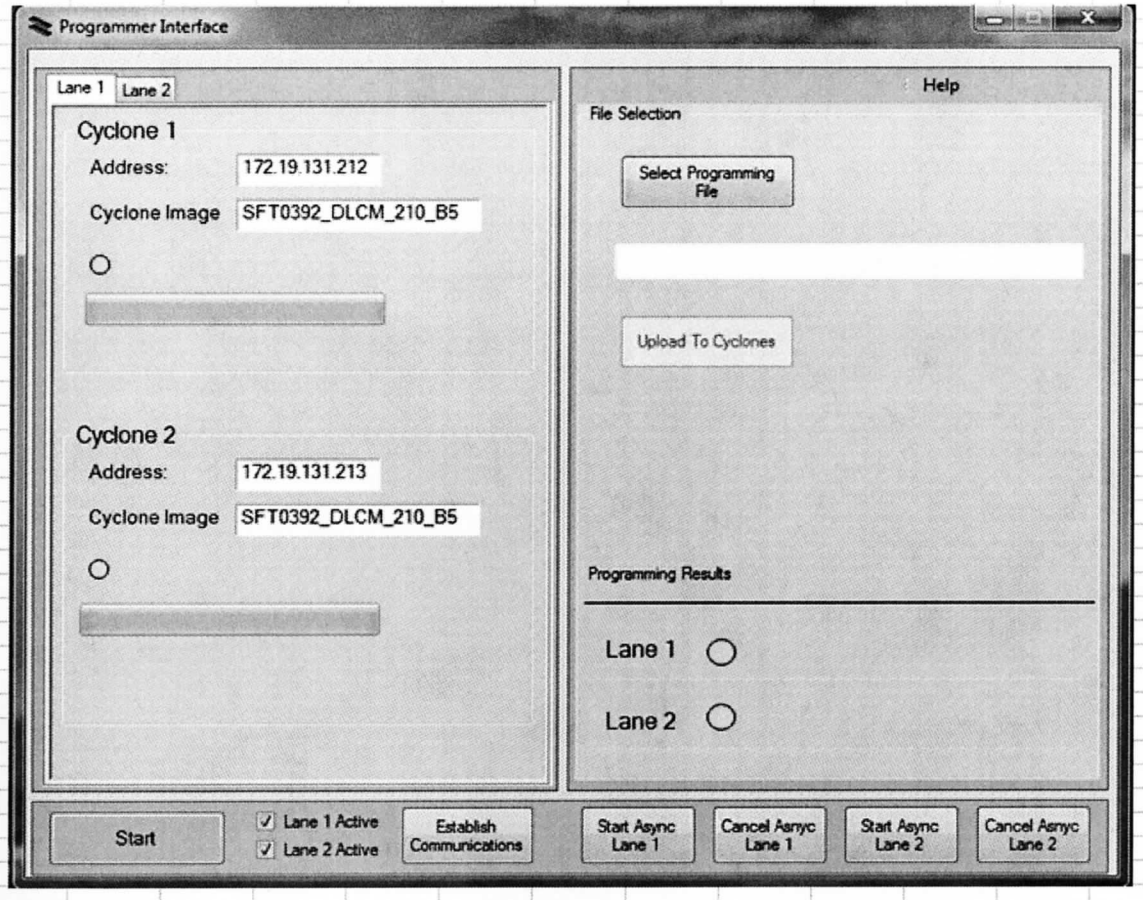
Select 'DLCM Programmer Interface' Shortcut from Desktop



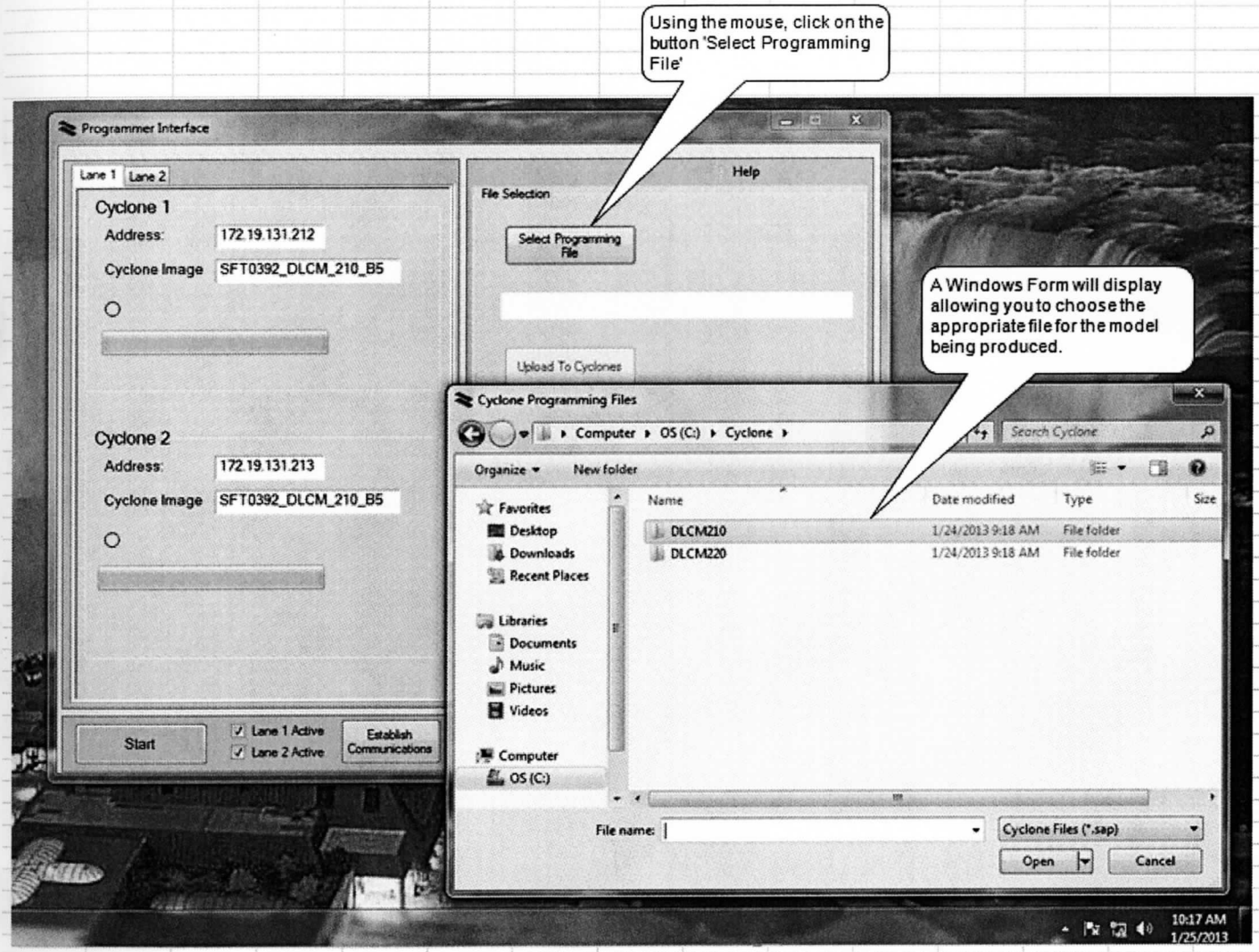
Or you can launch the application from the Start Menu



Once the application is opened you should see a form similar to the below image.

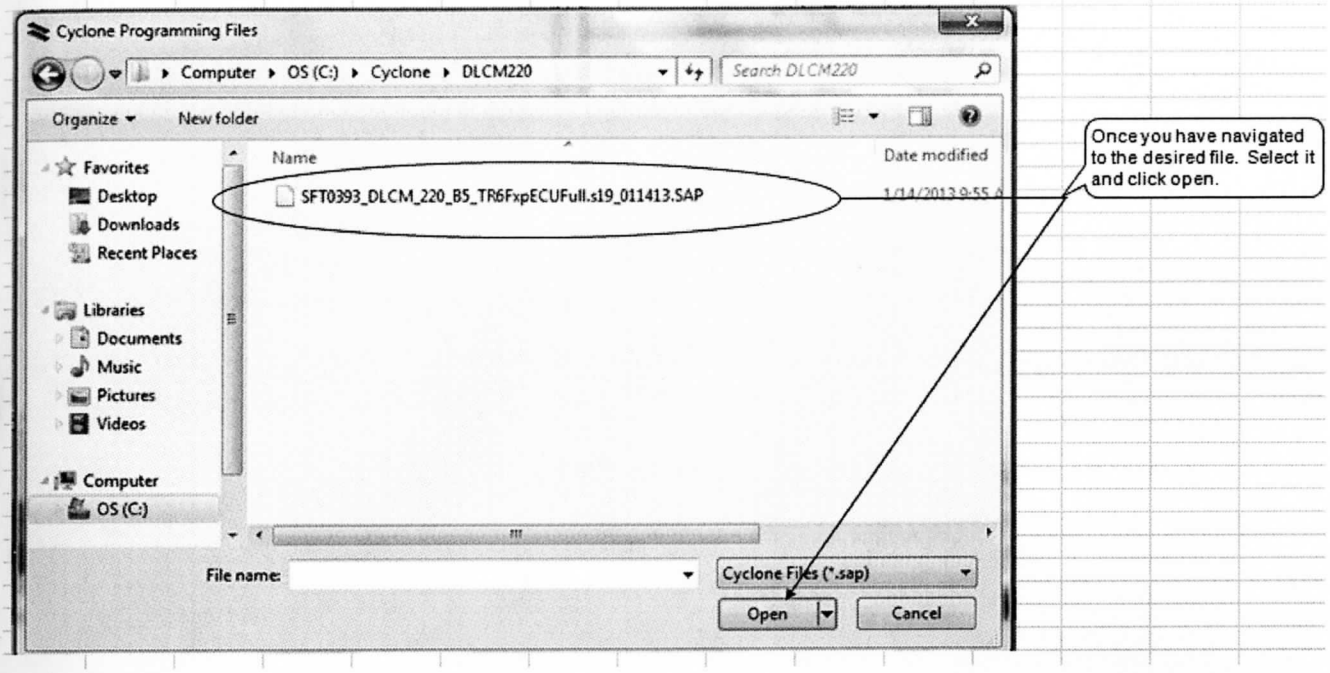


II: How To Select a programming file and upload to the Cyclone Pro Units

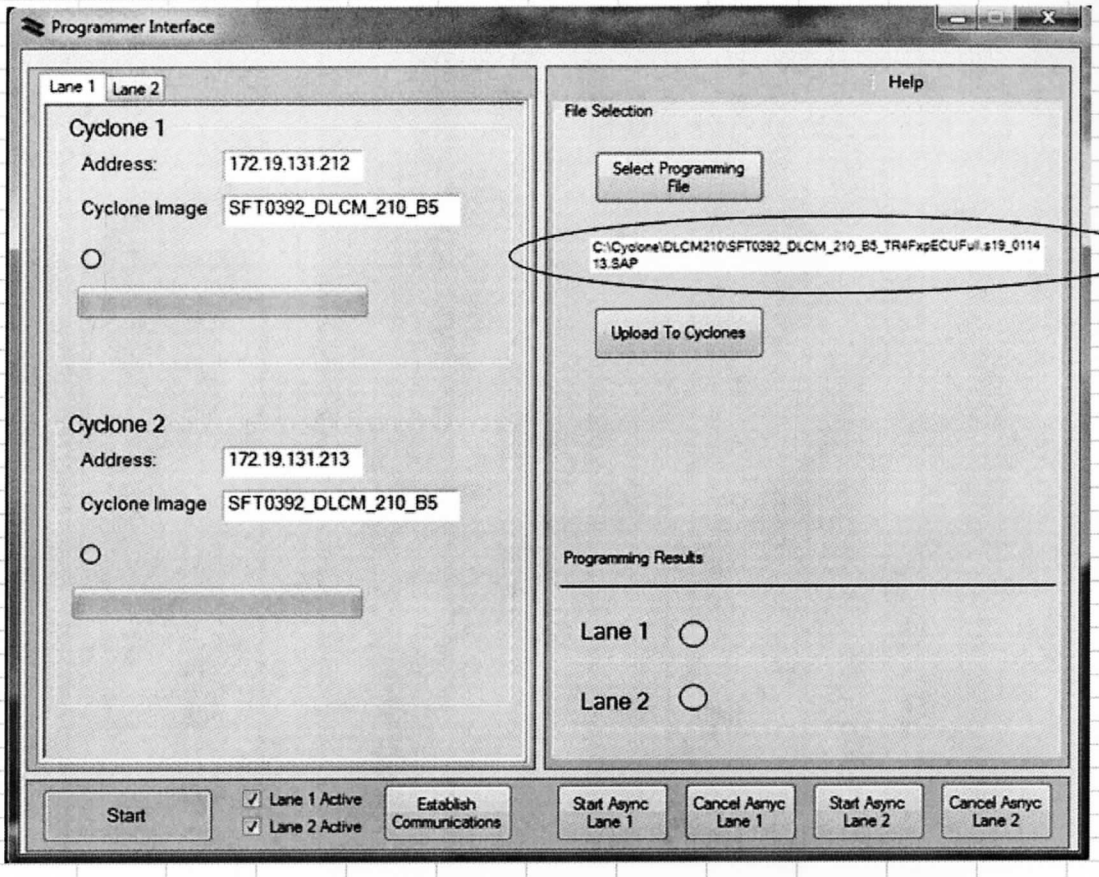


*Note ~ The directory path for file storage is 'C:\Cyclone*Model Type**.SAP file

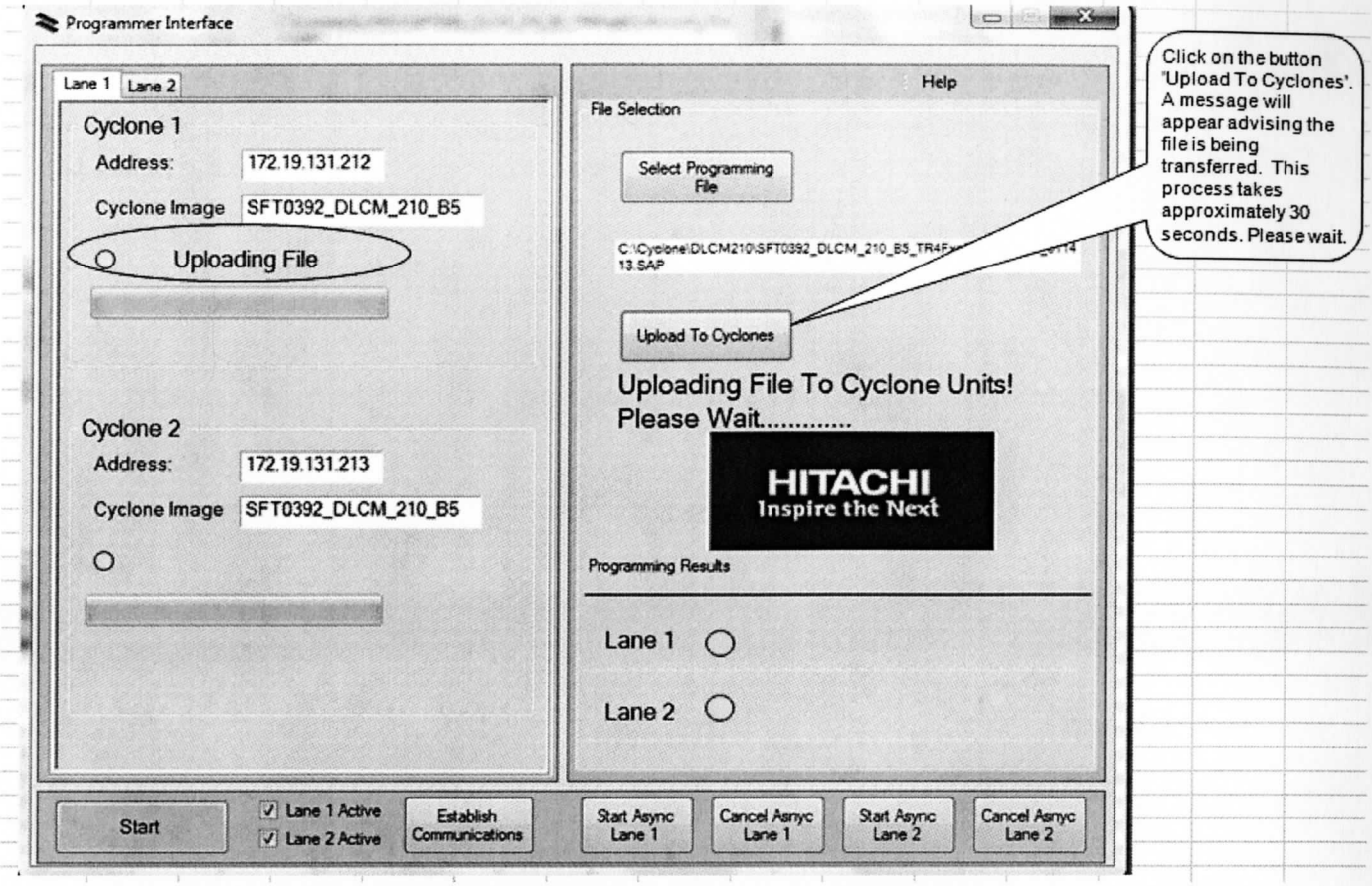
*Note ~ The directory path for file storage is 'C:\Cyclone*Model Type**.SAP file



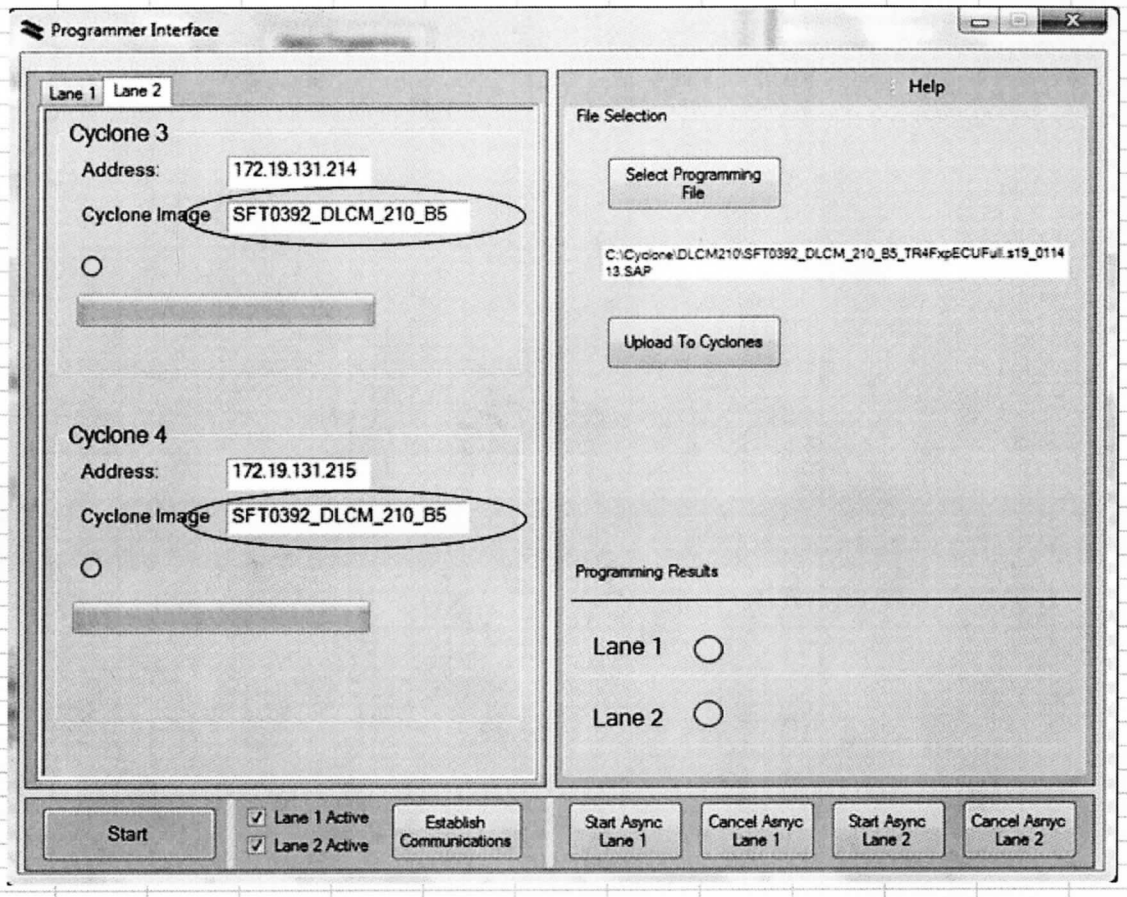
The file selected will be displayed in the label on the form. You now are ready to upload the programming file to the Cyclone Pro Units.



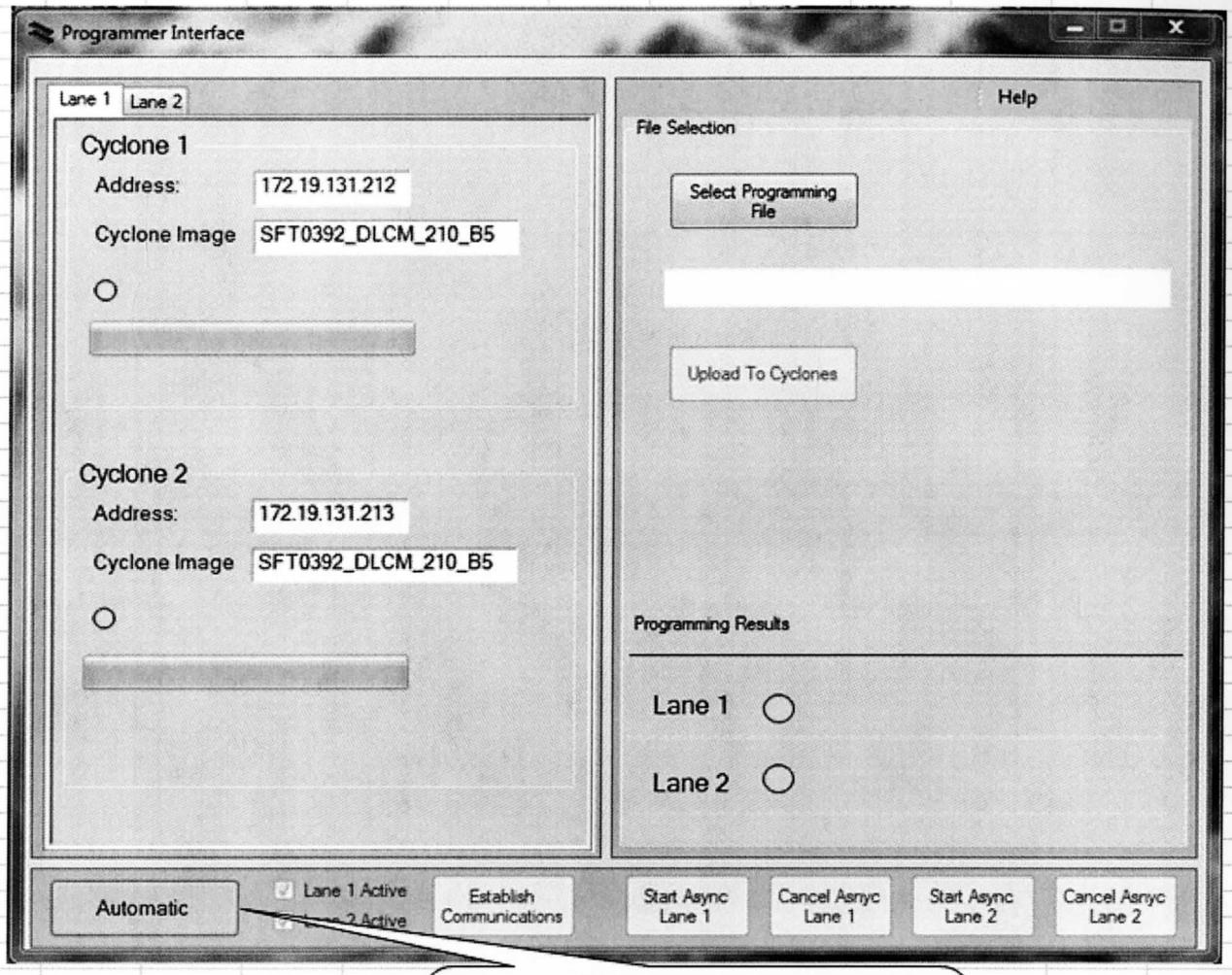
*Note ~ It is important to verify that all four Cyclone Pro Units are powered on. Ensure the Power LED on the units is on. If they are on proceed with the below steps. If they are not, check the machine controls are properly powered up.


Programmer Interface
Lane 1
Lane 2
Cyclone 1
Address:
172.19.131.212
Cyclone Image
SFT0392_DLCM_210_B5
Uploading File
Cyclone 2
Address:
172.19.131.213
Cyclone Image
SFT0392_DLCM_210_B5
Help
File Selection
Select Programming File
Upload To Cyclones
Uploading File To Cyclone Units!
Please Wait.....
HITACHI
Inspire the Next
Programming Results
Lane 1
Lane 2
Start
Lane 1 Active
Lane 2 Active
Establish Communications
Start Async Lane 1
Cancel Asnyc Lane 1
Start Async Lane 2
Cancel Asnyc Lane 2
Click on the button 'Upload To Cyclones'. A message will appear advising the file is being transferred. This process takes approximately 30 seconds. Please wait.

Once complete, the programming image stored on the Cyclone Pro Units will be displayed for all four devices.
*Note ~ Prior to equipment operation, verify the image file matches the desired model being produced.
Check all four devices (Lane 1 & Lane 2).

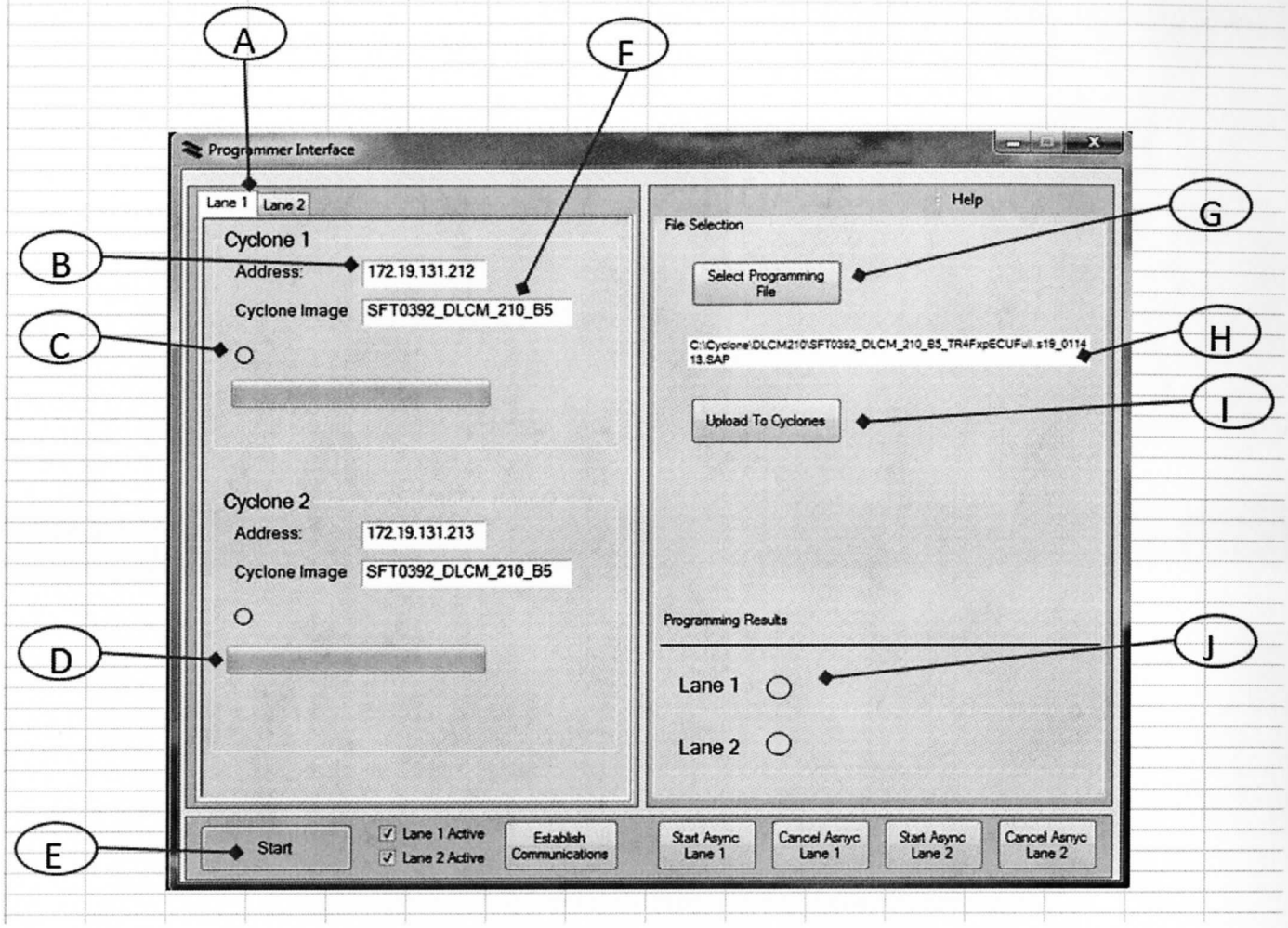


III: How to run the system in Automatic Mode



Once you are ready to run the system, click the start button. The button will display 'Automatic' and will turn to green. To stop Automatic mode, simply click the button once again and you will return to manual mode. *Note ~ you cannot stop an Auto-cycle until all programming processes are completed.

IV: Interface Overview



A: Display Tab; Lane 1 provides information for Cyclone units 1 & 2. Lane 2 provides information for Cyclone units 3 & 4.

B: The IP Address of the Cyclone Unit is displayed here.

C: A status indicator LED is provided for each Cyclone Pro Unit. Yellow = idle, Green = Pass, & Red = Fail.

D: A progress bar is used during the programming sequence which will update the user to the status of the current cycle.

E: This button will place the system in an automatic run mode. It will also return the system to a manual mode if selected while in automatic.

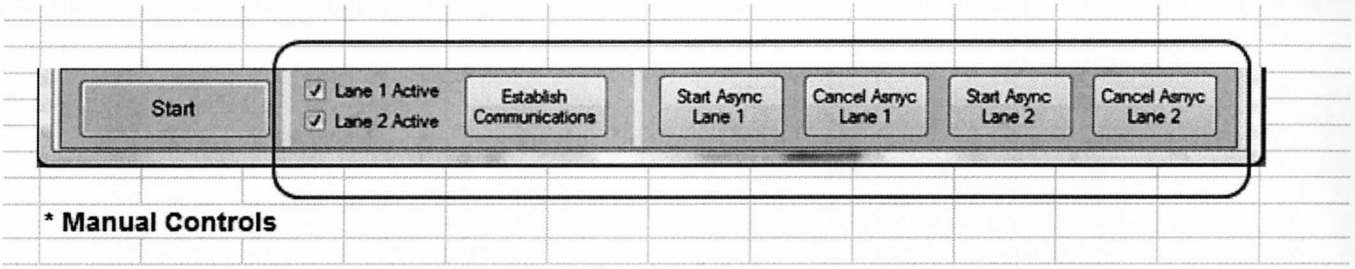
F: The current stored programming image on the Cyclone Pro units is displayed here.

G: This button is used to browse the Windows Directory for a Freescale programming file. (*.SAP)

H: Whenever a file is selected from the Windows directory, it is displayed here, and is ready to be uploaded into the Cyclone Pro Units.

I: Selecting this button will perform an 'upload sequence' sending the programming file into the memory of the Cyclone Pro Units.

J: The results for both programming lanes are displayed here. Text along with a status indicator LED is provided. Yellow = idle, Green = Pass, & Red = Fail.



Lane 1 Active / Lane 2 -- Active check boxes; If a lane is not in use for some reason, unselect the check box for that lane. This will allow some unnecessary processes to be deactivated while the lane is not in use.

Establish Communications -- This button will perform an initialization on all four Cyclone Pro Units. This is performed automatically upon the launching of the application so it is normally not required to be done.

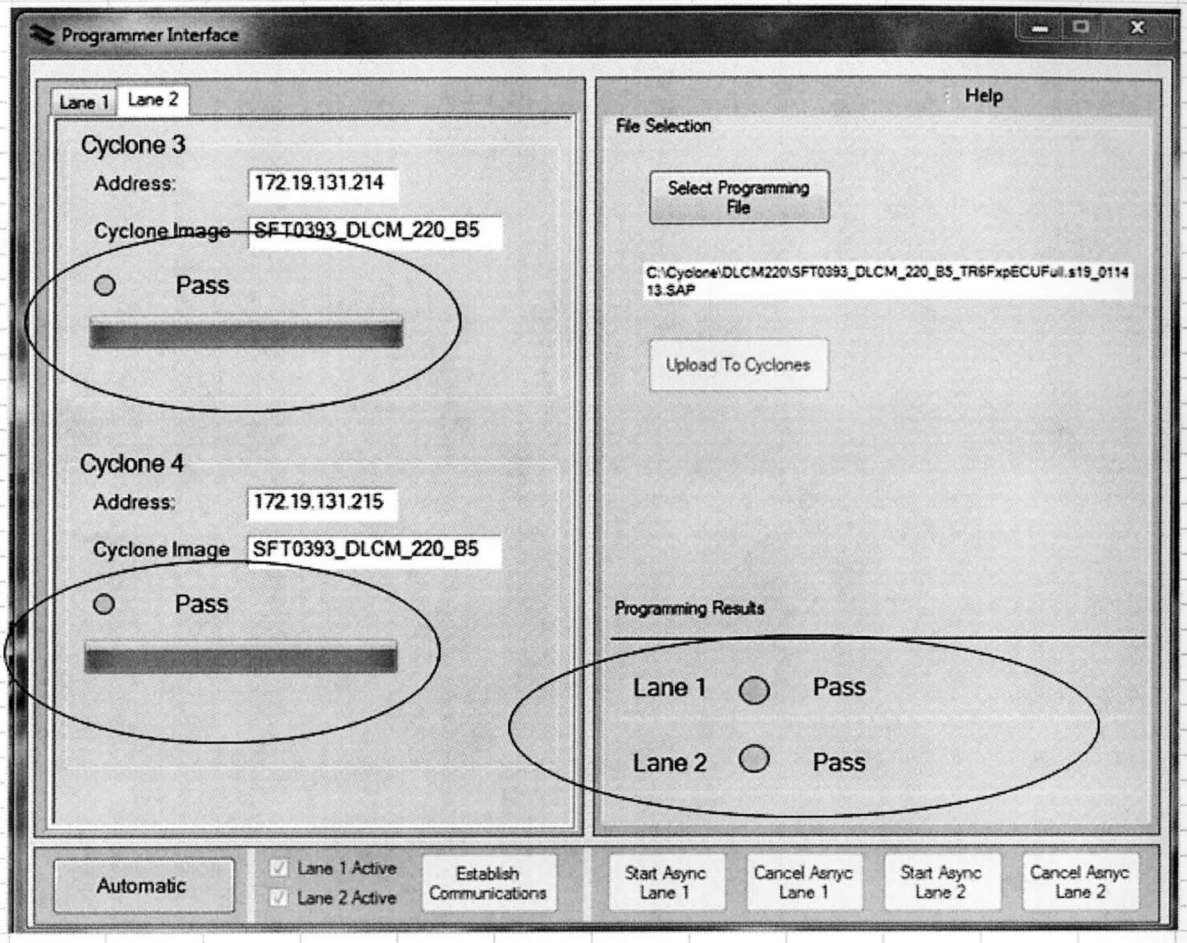
Start Async Lane 1 -- Initiates a programming sequence for lane 1. This is for debugging or manually reprogramming a panel. You will need to have a panel in place and the 12V supplied via the HMI to perform this.

Cancel Async Lane 1 -- Cancels (stops) the programming sequence for lane 1.

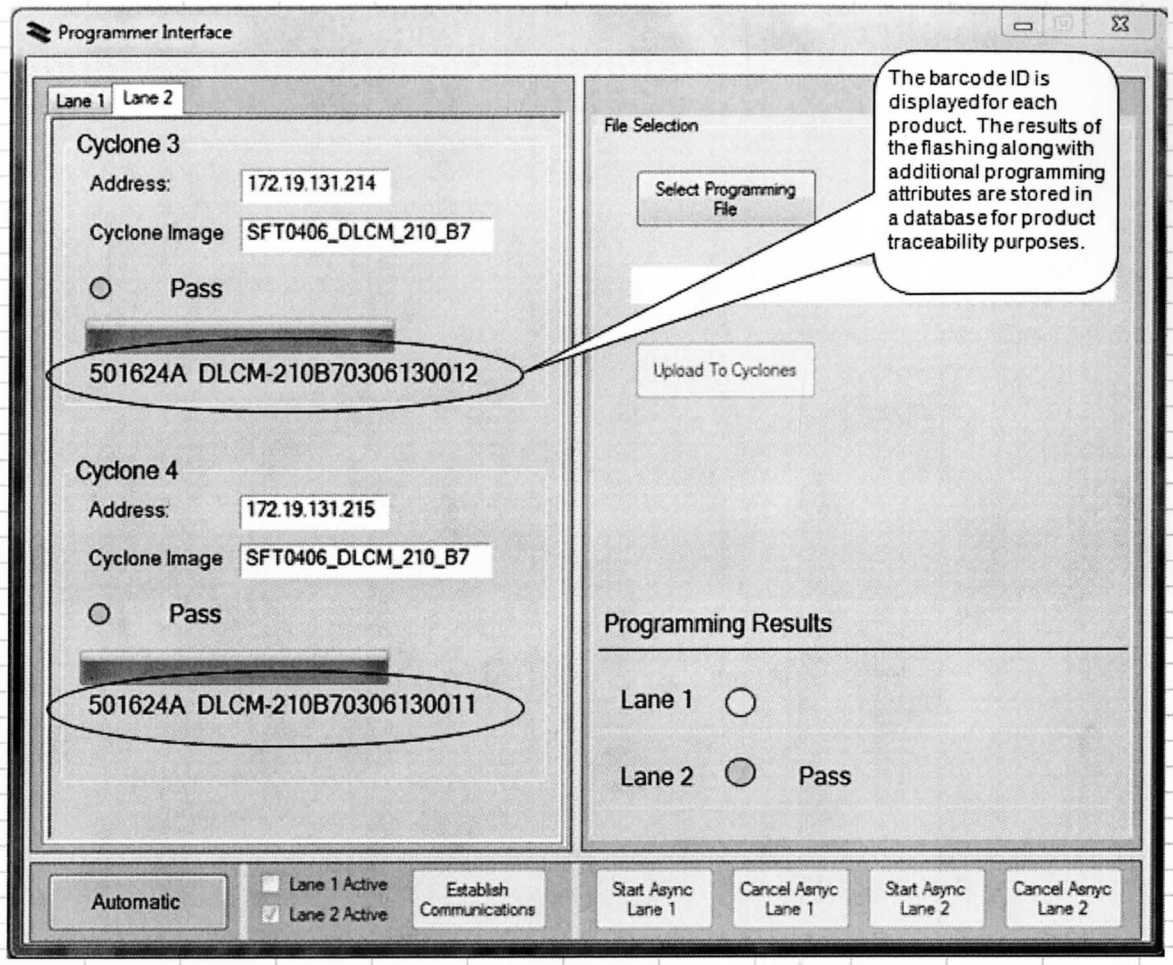
Start Async Lane 2 -- Initiates a programming sequence for lane 2. This is for debugging or manually reprogramming a panel. You will need to have a panel in place and the 12V supplied via the HMI to perform this.

Cancel Async Lane 2 -- Cancels (stops) the programming sequence for lane 2.

V: Results Display



VI: Traceability Data



* Note ~ In the event of a network disturbance or an inability to access the server, the results data and corresponding error log file are stored locally.

See the path below. *The results from the Record files will automatically be updated to the database once the connection issue has been resolved.

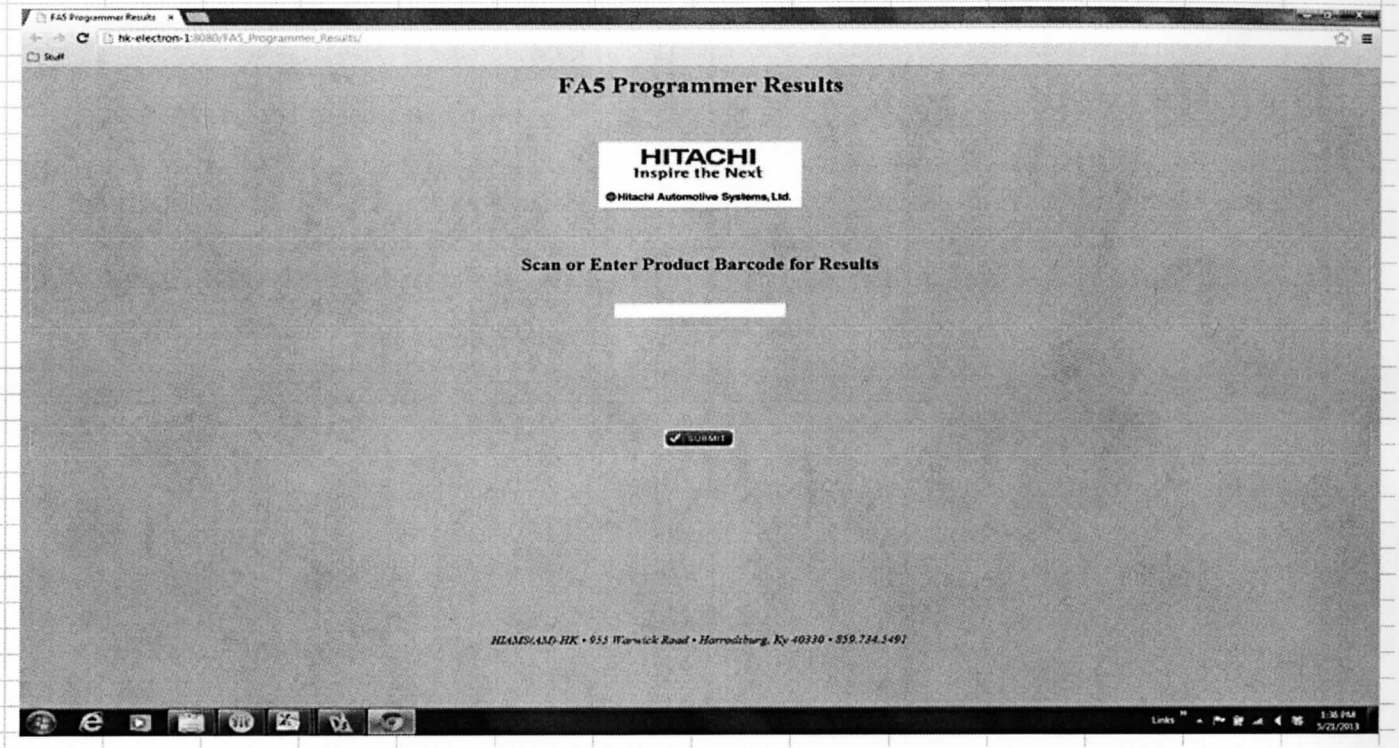
C:\Prg Data\db\Record*.log file
~ Error log file containing a description of the SQL connection issue along with a time stamp.

C:\Prg Data\db\Log*.log file
~ Record log file containing the pertinent results information necessary for database transactions.

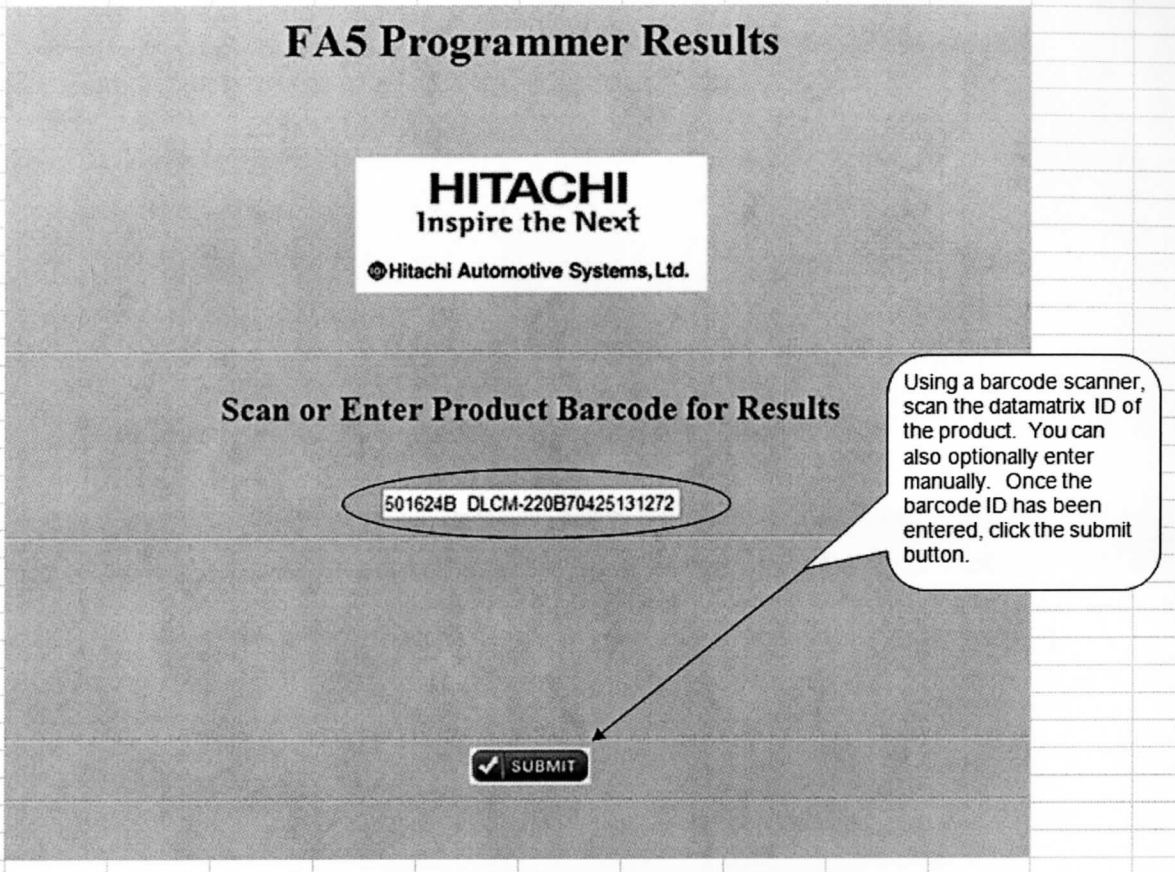
VII: Website Tutorial

Open your web browser, navigate to the following URL.

http://hk-electron-1:8080/FA5_Programer_Results/



Enter the product barcode ID (datamatix) and click the submit button



FA5 Programmer Results

HITACHI
Inspire the Next
Hitachi Automotive Systems, Ltd.

501624B DLCM-220B70425131272

Results			Machine Information	
Date / Time	Result	Cyclone Image	Lane Processed	Cyclone Unit Number
2013-04-25 12:15:46	P	SFT0407_DLCM_220_B7_TR6_FxpECU_Full_V09_01_08_022513	1	1

The results for the microprocessor flashing stations are provided to the user.

APPENDIX B: APPLICATION SOURCE CODE

Form1.vb

```
Option Explicit On
Option Strict On
Imports System.ComponentModel

Public Class Form1

'*************************************************************************
**************************
    'Programmer Interface for Cyclone Pro Modules.
    'Intended Use on DLCM products.. 2 up panel / 2 lane machine
    'Developed By Tilden May
    '03/2013
    'Utilizes 4 background worker threads as follows:
    'bckgrndWrk_ProSrv_Ln1  --> Polls Pro-Server Bits to start programming sequence
for Lane 1 <Cyclone 1 & 2>
    'bckgrndWrk_ProSrv_Ln2  --> Polls Pro-Server Bits to start programming sequence
for Lane 2 <Cyclone 3 & 4>
    'bckgrndWrk_Ln1  -->  Performs programming functionality with the Cyclone units
of Lane 1 <Cyclone 1 & 2>
    'bckgrndWrk_Ln2  -->  Performs programming functionality with the Cyclone units
of Lane 2 <Cyclone 3 & 4>

'*************************************************************************
**************************
    Dim tempCount As Integer
```

```
        Dim tempCount2 As Integer
        Dim Cyclone1_Image_Description As String
        Dim Cyclone2_Image_Description As String
        Dim Cyclone3_Image_Description As String
        Dim Cyclone4_Image_Description As String

        Dim Mode As Boolean  'handles Automatic or Manual mode <0 = manual, 1 =
automatic>
        Private Sub btn_file_select_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_file_select.Click
            Dim selectedFile As String

            opn_file_dialog.InitialDirectory = "C:\Cylone\"
            opn_file_dialog.FilterIndex = 1
            opn_file_dialog.Filter = "Cyclone Files (*.sap)|*.sap"

            If Me.opn_file_dialog.ShowDialog = Windows.Forms.DialogResult.OK Then
                selectedFile = opn_file_dialog.FileName
                lbl_selected_prg_file.Text = selectedFile.ToString
                lbl_selected_prg_file.Enabled = True
                btn_upload_cyclone.Enabled = True

            End If

        End Sub

        Public Function Upload_Cyclone(ByVal file_path As String) As Boolean
            Dim Success As Boolean = False

            Dim Cyclone1_EraseSuccess As Boolean
            Dim Cyclone1_ImageAddSuccess As Boolean
```

```
Dim Cyclone1_Image_Description As String
Dim Cyclone2_EraseSuccess As Boolean
Dim Cyclone2_ImageAddSuccess As Boolean
Dim Cyclone2_Image_Description As String
Dim Cyclone3_EraseSuccess As Boolean
Dim Cyclone3_ImageAddSuccess As Boolean
Dim Cyclone3_Image_Description As String
Dim Cyclone4_EraseSuccess As Boolean
Dim Cyclone4_ImageAddSuccess As Boolean
Dim Cyclone4_Image_Description As String


lblUpload_Msg.Text = "Uploading File To Cyclone Units!  Please Wait....."
lblUpload_Msg.Visible = True

'Cyclone 1 *** Uploading Selected File / Verifying Image Success

Try

  lbl_status_Cyc1.Text = "Erasing Cyclone"
  Cyclone1_EraseSuccess =
erase_all_cyclone_images(cyclonepromaxhandle:=Cyclone1_Handle)  'erase current image
from Cyclone

  If Cyclone1_EraseSuccess = False Then
    lbl_status_Cyc1.Text = "Error Erasing Image"
  End If

  If Cyclone1_EraseSuccess = True Then
    lbl_status_Cyc1.Text = "Uploading File Image"
```

```
                Cyclone1_ImageAddSuccess =
CBool(add_image_to_cyclone(cyclonepromaxhandle:=Cyclone1_Handle, aFile:=file_path))
'Upload selected file to Cyclone Unit

                If Cyclone1_ImageAddSuccess = False Then
                  lbl_status_Cyc1.Text = "Error Adding Image to Cyclone"
                ElseIf Cyclone1_ImageAddSuccess = True Then
                  lbl_status_Cyc1.Text = "Verifying Cyclone Image"
                  Cyclone1_Image_Description =
get_image_description(Cyclone1_Handle, 1) 'Read back current image from Cyclone
                  lbl_cyclone_image1.Text = Cyclone1_Image_Description.Substring(0,
19) 'Display the current image on form
                  lbl_status_Cyc1.Text = "" 'Clear status updates
                End If

              End If

            Catch ex As Exception
              Cyclone1_ImageAddSuccess = False
              lbl_status_Cyc1.Text = ""
            End Try

            'Cyclone 2 *** Uploading Selected File / Verifying Image Success
            Try

              lbl_status_Cyc2.Text = "Erasing Cyclone"
              Cyclone2_EraseSuccess =
erase_all_cyclone_images(cyclonepromaxhandle:=Cyclone2_Handle)  'erase current image
from Cyclone

              If Cyclone2_EraseSuccess = False Then
```

```
                lbl_status_Cyc2.Text = "Error Erasing Image"
            End If

            If Cyclone2_EraseSuccess = True Then
                lbl_status_Cyc2.Text = "Uploading File Image"
                Cyclone2_ImageAddSuccess =
CBool(add_image_to_cyclone(cyclonepromaxhandle:=Cyclone2_Handle, aFile:=file_path))
'Upload selected file to Cyclone Unit

                If Cyclone2_ImageAddSuccess = False Then
                    lbl_status_Cyc2.Text = "Error Adding Image to Cyclone"
                ElseIf Cyclone2_ImageAddSuccess = True Then
                    lbl_status_Cyc2.Text = "Verifying Cyclone Image"
                    Cyclone2_Image_Description =
get_image_description(Cyclone2_Handle, 1) 'Read back current image from Cyclone
                    lbl_cyclone_image2.Text = Cyclone2_Image_Description.Substring(0,
19)  'Display the current image on form
                    lbl_status_Cyc2.Text = "" 'Clear status updates
                End If

            End If

        Catch ex As Exception
            Cyclone2_ImageAddSuccess = False
            lbl_status_Cyc2.Text = ""
        End Try


        'Cyclone 3 *** Uploading Selected File / Verifying Image Success
        Try
```

```
                lbl_status_Cyc3.Text = "Erasing Cyclone"
                Cyclone3_EraseSuccess =
erase_all_cyclone_images(cyclonepromaxhandle:=Cyclone3_Handle)  'erase current image
from Cyclone

                If Cyclone3_EraseSuccess = False Then
                  lbl_status_Cyc3.Text = "Error Erasing Image"
                End If

                If Cyclone3_EraseSuccess = True Then
                  lbl_status_Cyc3.Text = "Uploading File Image"
                  Cyclone3_ImageAddSuccess =
CBool(add_image_to_cyclone(cyclonepromaxhandle:=Cyclone3_Handle, aFile:=file_path))
'Upload selected file to Cyclone Unit

                  If Cyclone3_ImageAddSuccess = False Then
                    lbl_status_Cyc3.Text = "Error Adding Image to Cyclone"
                  ElseIf Cyclone3_ImageAddSuccess = True Then
                    lbl_status_Cyc3.Text = "Verifying Cyclone Image"
                    Cyclone3_Image_Description =
get_image_description(Cyclone3_Handle, 1) 'Read back current image from Cyclone
                    lbl_cyclone_image3.Text = Cyclone3_Image_Description.Substring(0,
19)  'Display the current image on form
                    lbl_status_Cyc3.Text = "" 'Clear status updates
                  End If

                End If
              Catch ex As Exception
                Cyclone3_ImageAddSuccess = False
                lbl_status_Cyc3.Text = ""
              End Try
```

```
'Cyclone 4 *** Uploading Selected File / Verifying Image Success
Try
  lbl_status_Cyc4.Text = "Erasing Cyclone"
  Cyclone4_EraseSuccess =
erase_all_cyclone_images(cyclonepromaxhandle:=Cyclone4_Handle)  'erase current image
from Cyclone

If Cyclone4_EraseSuccess = False Then
  lbl_status_Cyc4.Text = "Error Erasing Image"
End If

If Cyclone4_EraseSuccess = True Then
  lbl_status_Cyc4.Text = "Uploading File Image"
  Cyclone4_ImageAddSuccess =
CBool(add_image_to_cyclone(cyclonepromaxhandle:=Cyclone4_Handle, aFile:=file_path))
'Upload selected file to Cyclone Unit

  If Cyclone4_ImageAddSuccess = False Then
    lbl_status_Cyc4.Text = "Error Adding Image to Cyclone"
  ElseIf Cyclone4_ImageAddSuccess = True Then
    lbl_status_Cyc4.Text = "Verifying Cyclone Image"
    Cyclone4_Image_Description =
get_image_description(Cyclone4_Handle, 1) 'Read back current image from Cyclone
    lbl_cyclone_image4.Text = Cyclone4_Image_Description.Substring(0,
19)  'Display the current image on form
    lbl_status_Cyc4.Text = "" 'Clear status updates
  End If

End If
```

```
            Catch ex As Exception
                Cyclone4_ImageAddSuccess = False
                lbl_status_Cyc4.Text = ""
            End Try

            lblUpload_Msg.Text = String.Empty
            lblUpload_Msg.Visible = False

            If Cyclone1_ImageAddSuccess And Cyclone2_ImageAddSuccess And
Cyclone3_ImageAddSuccess And Cyclone4_ImageAddSuccess Then
                Success = True
            End If

            Return Success

        End Function

        Private Sub btn_upload_cyclone_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btn_upload_cyclone.Click

            Dim selectedFile As String
            Dim Upload_Result As Boolean = False

            If lbl_selected_prg_file.Text <> String.Empty Then
                selectedFile = lbl_selected_prg_file.Text.Trim

                prg_bar1.Value = 0
                prg_bar2.Value = 0
                prg_bar3.Value = 0
                prg_bar4.Value = 0
```

```
            Upload_Result = Upload_Cyclone(selectedFile)

            If Not Upload_Result Then
              MessageBox.Show("There was an failure while uploading to the Cyclones.
Please verify!", "Alert")

            End If
          Else : MessageBox.Show("You must first select a Freescale programming file
(SAP).", "Alert")
          End If



        End Sub


        Private Function Establish_Communications_Cyclones(ByVal Mode As Boolean,
ByVal LaneChoice As Integer) As Boolean 'Mode F = Startup, T = Each Cycle; LaneChoice 1
= Lane 1, 2 = Lane 2

          'Dim Cyclone1_Image_Description As String
          'Dim Cyclone2_Image_Description As String
          'Dim Cyclone3_Image_Description As String
          'Dim Cyclone4_Image_Description As String
          Dim Connect_Status As Boolean = True

          If Not Mode Then
            'Upon Initial Startup Only  *** Establish a connection to all units
            Try
              Cyclone1_Handle = connect_to_cyclonepromax(port_type:=6,
identifier_type:=1, port_identifier:=Cyclone1_IP)
```

```
                lbl_IP_1.Text = Cyclone1_IP
                If Cyclone1_Handle = 0 Then
                    Connect_Status = False
                    MessageBox.Show("Unable to connect to Cyclone 1 <" + Cyclone1_IP +
"> !" + vbCrLf + "Please check device!", "Connectivity Error!", MessageBoxButtons.OK)
                Else

                End If

                Cyclone2_Handle = connect_to_cyclonepromax(port_type:=6,
identifier_type:=1, port_identifier:=Cyclone2_IP)
                lbl_IP_2.Text = Cyclone2_IP
                If Cyclone2_Handle = 0 Then
                    Connect_Status = False
                    MessageBox.Show("Unable to connect to Cyclone 2 <" + Cyclone2_IP +
"> !" + vbCrLf + "Please check device!", "Connectivity Error!", MessageBoxButtons.OK)
                Else

                End If

                Cyclone3_Handle = connect_to_cyclonepromax(port_type:=6,
identifier_type:=1, port_identifier:=Cyclone3_IP)
                lbl_IP_3.Text = Cyclone3_IP
                If Cyclone3_Handle = 0 Then
                    Connect_Status = False
                    MessageBox.Show("Unable to connect to Cyclone 3  <" + Cyclone3_IP +
"> !" + vbCrLf + "Please check device!", "Connectivity Error!", MessageBoxButtons.OK)
                Else

                End If
```

```
                Cyclone4_Handle = connect_to_cyclonepromax(port_type:=6,
identifier_type:=1, port_identifier:=Cyclone4_IP)
                lbl_IP_4.Text = Cyclone4_IP
                If Cyclone4_Handle = 0 Then
                    Connect_Status = False
                    MessageBox.Show("Unable to connect to Cyclone 4  <" + Cyclone4_IP +
"> !" + vbCrLf + "Please check device!", "Connectivity Error!", MessageBoxButtons.OK)
                Else

                End If

            Catch ex As Exception
                Connect_Status = False
                MessageBox.Show("An error occurred connecting to the Cyclones.  Verify
connectivity & power.", "Connection Error!", MessageBoxButtons.OK)
            End Try


        Else

            'Evaluate Each cycle if a connection is necessary.  Only if the units have been
reset (failure)
            Select Case LaneChoice

                Case 1
                    If Cyclone1_Handle <> 0 Then
                        Cyclone1_Handle = connect_to_cyclonepromax(port_type:=6,
identifier_type:=1, port_identifier:=Cyclone1_IP)
                    End If

                    If Cyclone2_Handle <> 0 Then
```

```
                    Cyclone2_Handle = connect_to_cyclonepromax(port_type:=6,
identifier_type:=1, port_identifier:=Cyclone2_IP)
                End If


            Case 2
                If Cyclone3_Handle <> 0 Then
                    Cyclone3_Handle = connect_to_cyclonepromax(port_type:=6,
identifier_type:=1, port_identifier:=Cyclone3_IP)
                End If

                If Cyclone4_Handle <> 0 Then
                    Cyclone4_Handle = connect_to_cyclonepromax(port_type:=6,
identifier_type:=1, port_identifier:=Cyclone4_IP)
                End If

            End Select

        End If

        Return Connect_Status


    End Function

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        Call CheckProcessInstance() 'Assure only one instance of app is in process

        bckgrndWrk_Ln1.WorkerReportsProgress = True
```

```
        bckgrndWrk_Ln2.WorkerReportsProgress = True
        bckgrndWrk_Ln1.WorkerSupportsCancellation = True
        bckgrndWrk_Ln2.WorkerSupportsCancellation = True
        lbl_results_lane1.Text = ""
        lbl_results_lane2.Text = ""

        Dim ProServerCheck As Boolean

        ProServerCheck = Write_ProServer(True, 1, True, 0, True, 0, True, 0, True, 0,
True, 0, True, 0) 'Update ProServer Bits <M900; application active>
        If ProServerCheck Then
            Me.Show()
            Call data_Cleanse()
            If Establish_Communications_Cyclones(False, 1) = True Then
                Call CycloneImageDescription()
            End If

        Else
            MessageBox.Show("An error occurred while communicating with
ProServer!", "Error!", MessageBoxButtons.OK)

        End If

        chkbx_ln1.Checked = True
        chkbx_ln2.Checked = True
        dm_ID_1.Visible = False
        dm_ID_2.Visible = False
        dm_ID_3.Visible = False
        dm_ID_4.Visible = False
```

```
        End Sub

        Private Sub CheckProcessInstance() 'Procedure for determining if application has a
process started or not

            Dim Proc() As Process

            'Determine the full name of the current process
            Dim ModuleName, ProcessName As String
            ModuleName = Process.GetCurrentProcess.MainModule.ModuleName
            ProcessName = System.IO.Path.GetFileNameWithoutExtension(ModuleName)

            'Find all processes with this name
            Proc = Process.GetProcessesByName(ProcessName)
            If Proc.Length > 1 Then
               'There is more than one process with this name running
               'Therefore force the application to end
               MessageBox.Show("This application is already running!", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
               Me.Close()

            End If

        End Sub

        Private Sub Form1_FormClosing(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
            Try
               Dim ProServerCheck As Boolean
               ProServerCheck = Write_ProServer(True, 0, True, 0, True, 0, True, 0, True, 0,
True, 0, True, 0) 'Update ProServer Bits <M900; application active>
```

```
        Call Disconnect_Cyclones_Lane1()
        Call Disconnect_Cyclones_Lane2()

    Catch ex As Exception

    End Try


End Sub

Private Sub CycloneImageDescription()

    Cyclone1_Image_Description = String.Empty
    Cyclone2_Image_Description = String.Empty
    Cyclone3_Image_Description = String.Empty
    Cyclone4_Image_Description = String.Empty


    Try
        Cyclone1_Image_Description = get_image_description(Cyclone1_Handle, 1)
        lbl_cyclone_image1.Text = Cyclone1_Image_Description.Substring(0, 19)
        Cyclone2_Image_Description = get_image_description(Cyclone2_Handle, 1)
        lbl_cyclone_image2.Text = Cyclone2_Image_Description.Substring(0, 19)
        Cyclone3_Image_Description = get_image_description(Cyclone3_Handle, 1)
        lbl_cyclone_image3.Text = Cyclone3_Image_Description.Substring(0, 19)
        Cyclone4_Image_Description = get_image_description(Cyclone4_Handle, 1)
        lbl_cyclone_image4.Text = Cyclone4_Image_Description.Substring(0, 19)

    Catch ex As Exception
        Dim errorMessage As String
```

```
            errorMessage = ex.Message.ToString
        End Try

    End Sub

    Private Sub bckgrndWrk_Ln1_DoWork(ByVal sender As System.Object, ByVal e
As System.ComponentModel.DoWorkEventArgs) Handles bckgrndWrk_Ln1.DoWork
        'Worker Thread initiated when a programming start request is received for lane 1

        Dim Cyclone1_Start As Boolean
        Dim Cyclone2_Start As Boolean
        Dim Units_Started As Boolean
        Dim Cycl_1_cnt As Integer = 0
        Dim Cycl_2_cnt As Integer = 0


        'Start Cyclone Programming Cycle
        Try
            While Not Cyclone1_Start And Cycl_1_cnt < 5
                Cyclone1_Start =
START_execute_all_commands(cyclonepromaxhandle:=Cyclone1_Handle, image_id:=1)
                Cycl_1_cnt = Cycl_1_cnt + 1
                If Not Cyclone1_Start Then
                    Threading.Thread.Sleep(0)
                End If
            End While

            While Not Cyclone2_Start And Cycl_2_cnt < 5

                Cyclone2_Start =
START_execute_all_commands(cyclonepromaxhandle:=Cyclone2_Handle, image_id:=1)
```

```
            Cycl_2_cnt = Cycl_2_cnt + 1
            If Not Cyclone2_Start Then
              Threading.Thread.Sleep(0)
            End If

          End While


        Catch ex As Exception

          bckgrndWrk_Ln2.CancelAsync() 'Cancel the thread if unable to start the
cyclones
          e.Cancel = True
        End Try

        If Cyclone1_Start And Cyclone2_Start Then
          Units_Started = True
        End If

        Dim Cyclone1_Status As UInteger
        Dim Cyclone2_Status As UInteger
        Dim intCnt_forError As Integer
        Dim ProgrammingComplete As Boolean = False
        Dim prgPercent As Integer
        Dim firstChk As Boolean
        'Check status of Cyclones to determine completion

        If Units_Started Then
          bckgrndWrk_Ln1.ReportProgress(1, "start")
          Try
            ' Threading.Thread.Sleep(1000)
```

```
                prgPercent = 5

                While Not ProgrammingComplete And
bckgrndWrk_Ln1.CancellationPending <> True

                    If intCnt_forError >= 24 Or Not firstChk Then
                        '0 returned = completed cycle
                        Threading.Thread.Sleep(0)
                        Cyclone1_Status =
check_STARTED_cyclonepromax_status(cyclonepromaxhandle:=Cyclone1_Handle)
                        Threading.Thread.Sleep(0)
                        Cyclone2_Status =
check_STARTED_cyclonepromax_status(cyclonepromaxhandle:=Cyclone2_Handle)
                    End If


                    If bckgrndWrk_Ln1.CancellationPending = True Then
                        e.Cancel = True
                    End If

                    If Cyclone1_Status = 0 And Cyclone2_Status = 0 Then
                        ProgrammingComplete = True
                    Else
                        firstChk = True
                        bckgrndWrk_Ln1.ReportProgress(prgPercent, "busy")
                        prgPercent = prgPercent + 4
                        Threading.Thread.Sleep(5000)

                        If intCnt_forError >= 31 Then   'In case of max # of cycles on loop;
consider a communication error occurred with Cyclone
                            ProgrammingComplete = True
```

```
                End If

            End If

            intCnt_forError = intCnt_forError + 1

        End While

    Catch ex As Exception
        bckgrndWrk_Ln1.CancelAsync() 'Cancel the thread if unable to start the
cyclones
        e.Cancel = True
    End Try

Else : e.Result = CStr(1) + CStr(1)  '0 = pass,  1 = fail  {Fail Lane if unable to
start programming sequence}
    bckgrndWrk_Ln1.CancelAsync() 'Cancel the thread if unable to start the
cyclones
    e.Cancel = True
End If

If Units_Started And ProgrammingComplete Then

    Dim Cyclone1_GetResult As UShort
    Dim Cyclone2_GetResult As UShort

    Try
        '**** If Cyclone Status remains at 1; after an extended amount of time, the
module is presumed to be unable to program
        '**** In this case, set the result to 1 <Fail>
```

```
                If Cyclone1_Status <> 1 Then
                  Threading.Thread.Sleep(0)

                  Cyclone1_GetResult =
get_last_error_code(cyclonepromaxhandle:=Cyclone1_Handle)
                Else
                  : Cyclone1_GetResult = 2

                End If

                If Cyclone2_Status <> 1 Then
                  Threading.Thread.Sleep(0)
                  Cyclone2_GetResult =
get_last_error_code(cyclonepromaxhandle:=Cyclone2_Handle)
                Else : Cyclone2_GetResult = 2
                End If


                e.Result = CStr(Cyclone1_GetResult) + CStr(Cyclone2_GetResult)  '0 =
pass,  1 = fail

              Catch ex As Exception
              End Try

            End If


            If bckgrndWrk_Ln1.CancellationPending = True Then
              e.Cancel = True
            End If
```

```
End Sub

Private Sub bckgrndWrk_Ln1_ProgressChanged(ByVal sender As System.Object,
ByVal e As System.ComponentModel.ProgressChangedEventArgs) Handles
bckgrndWrk_Ln1.ProgressChanged

    btn_upload_cyclone.Enabled = False 'disable uploading button during a
programming cycle
    btn_file_select.Enabled = False 'disable file selection while programming
    tempCount = tempCount + 1
    '  tempLabel_1.Text = CStr(tempCount)

    If dm_ID_1.Text <> dmCode_Ln1 Then
       dm_ID_1.Visible = True
       dm_ID_1.Text = dmCode_Ln1.Trim

    End If

    If dm_ID_2.Text <> dmCode_Ln1_2 Then
       dm_ID_2.Visible = True
       dm_ID_2.Text = dmCode_Ln1_2.Trim

    End If


    If prg_bar1.Value >= 95 Then 'Update the progress bar if it hasn't maxed out
       prg_bar1.Value = 95

    Else : prg_bar1.Value = e.ProgressPercentage
    End If
```

```
        If prg_bar2.Value >= 95 Then 'Update the progress bar if it hasn't maxed out
            prg_bar2.Value = 95

        Else : prg_bar2.Value = e.ProgressPercentage
        End If


        Dim startChk As Boolean
        startChk = Write_ProServer(False, 0, False, 0, False, 0, False, 0, False, 0, True, 1,
False, 0) 'Update ProServer Bits <M909; Lane1 programming started

        'Determine Cyclone Activity & Status
        Dim status As String = CStr(e.UserState)
        If status = "start" Then

            lbl_status_Cyc1.Text = "Erasing"
            lbl_status_Cyc2.Text = "Erasing"
        ElseIf status = "busy" Then
            lbl_status_Cyc1.Text = "Programming"
            lbl_status_Cyc2.Text = "Programming"
        End If

    End Sub

    Private Sub bckgrndWrk_Ln1_RunWorkerCompleted(ByVal sender As
System.Object, ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs)
Handles bckgrndWrk_Ln1.RunWorkerCompleted

        Dim Lane1_Results, resultsChk, boolReset1, boolReset2 As Boolean
        Dim result As String
```

```
Dim Cyc1_result_db, Cyc2_result_db As Char
Dim Cyc1_result, Cyc2_result As Integer
Cyc1_result = -1
Cyc2_result = -1

If e.Cancelled <> True Then
   result = e.Result.ToString  'get results for cyclones 1 & 2
   Cyc1_result = CInt(result.Substring(0, 1))
   Cyc2_result = CInt(result.Substring(1, 1))

   If Cyc1_result = 0 And Cyc2_result = 0 Then
      Lane1_Results = True

   Else
      If Cyc1_result = 2 Then
         boolReset1 = True
      End If

      If Cyc2_result = 2 Then
         boolReset2 = True
      End If
   End If


End If

'Handle GUI services
Select Case Cyc1_result
   Case 0
      lbl_status_Cyc1.Text = "Pass"
      Cyc1_result_db = CChar("P")
```

```
        ovl_inprogress_Cyc1.FillColor = Color.LightGreen
        prg_bar1.Value = 100

    Case 1
        lbl_status_Cyc1.Text = "FAIL"
        Cyc1_result_db = CChar("F")
        ovl_inprogress_Cyc1.FillColor = Color.Red
    Case Else
        lbl_status_Cyc1.Text = "FAIL"
        Cyc1_result_db = CChar("F")
        ovl_inprogress_Cyc1.FillColor = Color.Red
End Select

Select Case Cyc2_result
    Case 0
        lbl_status_Cyc2.Text = "Pass"
        Cyc2_result_db = CChar("P")
        ovl_inprogress_Cyc2.FillColor = Color.LightGreen
        prg_bar2.Value = 100
    Case 1
        lbl_status_Cyc2.Text = "FAIL"
        Cyc2_result_db = CChar("F")
        ovl_inprogress_Cyc2.FillColor = Color.Red
    Case Else
        lbl_status_Cyc2.Text = "FAIL"
        Cyc2_result_db = CChar("F")
        ovl_inprogress_Cyc2.FillColor = Color.Red

End Select
```

```
            Select Case Lane1_Results
              'Provide resutls to HMI via ProServer
              Case True
                resultsChk = Write_ProServer(False, 0, False, 0, True, 1, False, 0, False, 0,
False, 0, False, 0) 'Update ProServer Bits <M904; Lane1 passed programming
                ovl_lane1.FillColor = Color.LightGreen
                lbl_results_lane1.Text = "Pass"
              Case False
                resultsChk = Write_ProServer(False, 0, False, 0, False, 0, True, 1, False, 0,
False, 0, False, 0) 'Update ProServer Bits <M905; Lane1 failed programming
                ovl_lane1.FillColor = Color.Red
                lbl_results_lane1.Text = "Fail"
            End Select


            Call Db_Services(dmCode_Ln1, 1, Cyc1_result_db)  'Handle db services for the
results from Lane 1 / PCB 1
            Call Db_Services(dmCode_Ln1_2, 2, Cyc2_result_db) 'Handle db services for
the results from Lane 1 / PCB 2

            If boolReset1 Or boolReset2 Then

              Call Reset_Cyclones_Lane1(boolReset1, boolReset2)
              Threading.Thread.Sleep(1000) 'sleep the thead to allow the Cyclone Units time
to reboot
            End If


            If bckgrndWrk_Ln2.IsBusy = False Then
```

```
            btn_file_select.Enabled = True 'enable file selection while programming

        End If
        If bckgrndWrk_ProSrv_Ln1.IsBusy <> True Then
            Call bckgrndWrk_ProSrv_Ln1.RunWorkerAsync() 'Initiate the thread for
monitoring ProServer for a start signal request
        End If


        tempCount = 0

    End Sub

    Private Sub btn_Man_StartAsync_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btn_Man_StartAsync_Ln1.Click

        If chkbx_ln1.Checked And bckgrndWrk_Ln1.IsBusy <> True Then
            Call Reset_GUI_Idle_State(1)
            Call Establish_Communications_Cyclones(True, 1)
            bckgrndWrk_Ln1.RunWorkerAsync()  'Initiates the worker thread for
programming lane 1
        End If

    End Sub

    Private Sub bnt_Man_Cancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_Cncl_Ln1.Click

        If bckgrndWrk_Ln1.IsBusy = True Then
            bckgrndWrk_Ln1.CancelAsync()  'Cancels the worker thread for programming
lane 1
```

```
        End If


    End Sub

    Private Sub btn_Comm_Cyclones_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btn_Comm_Cyclones.Click
        'Establishes communication with the cyclones
        btn_Comm_Cyclones.Enabled = False
        Call Disconnect_Cyclones_Lane1() 'Disconnect Cyclone Units 1 / 2
        Call Disconnect_Cyclones_Lane2() 'Disconnect Cyclone Units 3 / 4
        'Clear all handles
        Cyclone1_Handle = 0
        Cyclone2_Handle = 0
        Cyclone3_Handle = 0
        Cyclone4_Handle = 0
        Call Establish_Communications_Cyclones(False, 1)
        btn_Comm_Cyclones.Enabled = True

    End Sub
    Private Sub Disable_Manual_Controls()

        chkbx_ln1.Enabled = False
        chkbx_ln2.Enabled = False
        btn_Comm_Cyclones.Enabled = False
        btn_Man_StartAsync_Ln1.Enabled = False
        btn_Man_StartAsync_Ln2.Enabled = False
        btn_Cncl_ln2.Enabled = False
        btn_Cncl_Ln1.Enabled = False
        btn_file_select.Enabled = False
        btn_upload_cyclone.Enabled = False
```

```
End Sub

Private Sub Enable_Manual_Controls()

    chkbx_ln1.Enabled = True
    chkbx_ln2.Enabled = True
    btn_Comm_Cyclones.Enabled = True
    btn_Man_StartAsync_Ln1.Enabled = True
    btn_Man_StartAsync_Ln2.Enabled = True
    btn_Cncl_ln2.Enabled = True
    btn_Cncl_Ln1.Enabled = True
    btn_file_select.Enabled = True
    btn_upload_cyclone.Enabled = True


End Sub

Private Sub btn_Start_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_Start.Click

    Select Case Mode
        Case False  'Currently in manual mode; request to start Automatic

            If bckgrndWrk_Ln1.IsBusy <> True And bckgrndWrk_Ln2.IsBusy <> True
Then 'Only permit if programming threads aren't active
                If chkbx_ln1.Checked = False And chkbx_ln2.Checked = False Then 'Do
not permit if both lanes are de-selected
                    MessageBox.Show("At least one programming lane must be selected!",
"Not So Fast.....", MessageBoxButtons.OK)
                Else
```

```
                Call Disable_Manual_Controls()
                btn_Start.Text = "Automatic"
                btn_Start.BackColor = Color.LightGreen
                Mode = True

                If chkbx_ln1.Checked Then
                    Call Reset_Cyclones_Lane1(True, True)
                    If bckgrndWrk_ProSrv_Ln1.IsBusy <> True Then
                        bckgrndWrk_ProSrv_Ln1.RunWorkerAsync()
                    End If

                End If

                If chkbx_ln2.Checked Then
                    Call Reset_Cyclones_Lane2(True, True)
                    If bckgrndWrk_ProSrv_Ln2.IsBusy <> True Then
                        bckgrndWrk_ProSrv_Ln2.RunWorkerAsync()
                    End If
                End If

            End If

        End If


    Case True   'Currently in automatic mode; requesting to place in Manual
        If bckgrndWrk_Ln1.IsBusy <> True And bckgrndWrk_Ln2.IsBusy <> True
Then

            Call Enable_Manual_Controls()
            Call bckgrndWrk_ProSrv_Ln1.CancelAsync()
            Call bckgrndWrk_ProSrv_Ln2.CancelAsync()
```

```
                btn_Start.Text = "Start"
                btn_Start.BackColor = Color.FromKnownColor(KnownColor.ScrollBar)
                Mode = False
                Call data_Cleanse()
            End If

        End Select

    End Sub

    Private Sub btn_Man_StartAsync_Ln2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btn_Man_StartAsync_Ln2.Click

        If chkbx_ln2.Checked And bckgrndWrk_Ln2.IsBusy <> True Then
            Call Reset_GUI_Idle_State(2)
            Call Establish_Communications_Cyclones(True, 2)
            bckgrndWrk_Ln2.RunWorkerAsync()  'Initiates the worker thread for
programming lane 2
        End If
    End Sub

    Private Sub btn_Cncl_ln2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_Cncl_ln2.Click
        If bckgrndWrk_Ln2.IsBusy = True Then
            bckgrndWrk_Ln2.CancelAsync()  'Cancels the thread for lane 2
        End If

    End Sub
    Private Sub bckgrndWrk_ProSrv_Ln1_DoWork(ByVal sender As System.Object,
ByVal e As System.ComponentModel.DoWorkEventArgs) Handles
bckgrndWrk_ProSrv_Ln1.DoWork
```

```
        'Thread handles monitoring inputs from ProServer for Lane 1 actions

        Dim myLane As Integer = 1
        Dim check_PrServ As Integer = 0
        Dim request_to_Prgm As Boolean

        While bckgrndWrk_ProSrv_Ln1.CancellationPending <> True And
request_to_Prgm = False

          check_PrServ = Read_ProServer(myLane)
          If check_PrServ >= 1 Then
            request_to_Prgm = True
          Else
            Threading.Thread.Sleep(1000)
          End If

        End While

        e.Result = request_to_Prgm

    End Sub

    Private Sub bckgrndWrk_ProSrv_Ln1_ProgressChanged(ByVal sender As Object,
ByVal e As System.ComponentModel.ProgressChangedEventArgs) Handles
bckgrndWrk_ProSrv_Ln1.ProgressChanged

    End Sub

    Private Sub bckgrndWrk_ProSrv_Ln1_RunWorkerCompleted(ByVal sender As
System.Object, ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs)
Handles bckgrndWrk_ProSrv_Ln1.RunWorkerCompleted
```

```
        If CBool(e.Result) = True Then
            Call Reset_GUI_Idle_State(1)
            Call Establish_Communications_Cyclones(True, 1)
            Call bckgrndWrk_Ln1.RunWorkerAsync() 'Start the worker thread for
programming lane 1

        End If

    End Sub

    Private Sub Reset_GUI_Idle_State(ByVal Lane_Check As Integer)

        Select Case Lane_Check

            Case 1

                lbl_results_lane1.Text = ""
                lbl_status_Cyc1.Text = ""
                lbl_status_Cyc2.Text = ""
                ovl_inprogress_Cyc1.FillColor = Color.Yellow
                ovl_inprogress_Cyc2.FillColor = Color.Yellow
                ovl_lane1.FillColor = Color.Yellow
                prg_bar1.Value = 0
                prg_bar2.Value = 0


            Case 2

                lbl_results_lane2.Text = ""
                lbl_status_Cyc3.Text = ""
```

```
            lbl_status_Cyc4.Text = ""
            ovl_inprogress_Cyc3.FillColor = Color.Yellow
            ovl_Inprogress_Cyc4.FillColor = Color.Yellow
            ovl_lane2.FillColor = Color.Yellow
            prg_bar3.Value = 0
            prg_bar4.Value = 0

        End Select

    End Sub

    Private Sub AboutToolStripMenuItem1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles AboutToolStripMenuItem1.Click
        Dim helpForm As New Help_About
        helpForm.ShowDialog()
    End Sub

    Private Sub bckgrndWrk_ProSrv_Ln2_DoWork(ByVal sender As System.Object,
ByVal e As System.ComponentModel.DoWorkEventArgs) Handles
bckgrndWrk_ProSrv_Ln2.DoWork

        'Thread handles monitoring inputs from ProServer for Lane 2 actions

        Dim myLane As Integer = 2
        Dim check_PrServ As Integer = 0
        Dim request_to_Prgm As Boolean

        While bckgrndWrk_ProSrv_Ln2.CancellationPending <> True And
request_to_Prgm = False

            check_PrServ = Read_ProServer(myLane)
```

```
            If check_PrServ >= 1 Then
               request_to_Prgm = True
            Else
               Threading.Thread.Sleep(1000)
            End If

         End While

         e.Result = request_to_Prgm

      End Sub

      Private Sub bckgrndWrk_ProSrv_Ln2_RunWorkerCompleted(ByVal sender As
System.Object, ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs)
Handles bckgrndWrk_ProSrv_Ln2.RunWorkerCompleted

         If CBool(e.Result) = True Then
            Call Reset_GUI_Idle_State(2)
            Call Establish_Communications_Cyclones(True, 2)
            Call bckgrndWrk_Ln2.RunWorkerAsync() 'Start the worker thread for
programming lane 1

         End If
      End Sub

      Private Sub bckgrndWrk_Ln2_DoWork(ByVal sender As System.Object, ByVal e
As System.ComponentModel.DoWorkEventArgs) Handles bckgrndWrk_Ln2.DoWork

         Dim Cyclone3_Start As Boolean
         Dim Cyclone4_Start As Boolean
         Dim Units_Started As Boolean
```

```
Dim Cycl_3_cnt As Integer = 0
Dim Cycl_4_cnt As Integer = 0


'Start Cyclone Programming Cycle
Try
  While Not Cyclone3_Start And Cycl_3_cnt < 3
    Cyclone3_Start =
START_execute_all_commands(cyclonepromaxhandle:=Cyclone3_Handle, image_id:=1)
    Cycl_3_cnt = Cycl_3_cnt + 1
    If Not Cyclone3_Start Then
      Threading.Thread.Sleep(0)
    End If
  End While

  While Not Cyclone4_Start And Cycl_4_cnt < 3

    Cyclone4_Start =
START_execute_all_commands(cyclonepromaxhandle:=Cyclone4_Handle, image_id:=1)
    Cycl_4_cnt = Cycl_4_cnt + 1
    If Not Cyclone4_Start Then
      Threading.Thread.Sleep(0)
    End If

  End While


  If Cyclone3_Start And Cyclone4_Start Then
    Units_Started = True
  End If
```

```
            Catch ex As Exception

               bckgrndWrk_Ln2.CancelAsync() 'Cancel the thread if unable to start the
cyclones
               e.Cancel = True
            End Try

            Dim Cyclone3_Status As UInteger
            Dim Cyclone4_Status As UInteger
            Dim intCnt_forError As Integer
            Dim ProgrammingComplete As Boolean = False
            Dim prgPercent As Integer
            Dim firstChk As Boolean

            'Check status of Cyclones to determine completion
            If Units_Started Then
               bckgrndWrk_Ln2.ReportProgress(1, "start")
               Try
                  'Threading.Thread.Sleep(1000)
                  prgPercent = 5

                  While Not ProgrammingComplete And
bckgrndWrk_Ln2.CancellationPending <> True

                     '0 returned = completed cycle, 1 = busy
                     If intCnt_forError >= 24 Or Not firstChk Then
                        Threading.Thread.Sleep(0)
                        Cyclone3_Status =
check_STARTED_cyclonepromax_status(cyclonepromaxhandle:=Cyclone3_Handle)
                        Threading.Thread.Sleep(0)
```

```
                    Cyclone4_Status =
check_STARTED_cyclonepromax_status(cyclonepromaxhandle:=Cyclone4_Handle)
                End If

                If bckgrndWrk_Ln2.CancellationPending = True Then
                    e.Cancel = True
                End If

                If Cyclone3_Status = 0 And Cyclone4_Status = 0 Then
                    ProgrammingComplete = True
                Else
                    firstChk = True
                    bckgrndWrk_Ln2.ReportProgress(prgPercent, "busy")
                    prgPercent = prgPercent + 4
                    Threading.Thread.Sleep(5000)

                    If intCnt_forError >= 31 Then   'In case of max cycles of loop; consider
a communication error occurred with Cyclone
                        ProgrammingComplete = True
                    End If

                End If

                intCnt_forError = intCnt_forError + 1

            End While

        Catch ex As Exception
            bckgrndWrk_Ln2.CancelAsync() 'Cancel the thread if unable to start the
cyclones
            e.Cancel = True
```

```
            End Try

        Else : e.Result = CStr(1) + CStr(1)  '0 = pass,  1 = fail  {Fail Lane if unable to
start programming sequence}

            bckgrndWrk_Ln2.CancelAsync() 'Cancel the thread if unable to start the
cyclones
            e.Cancel = True
        End If

        If Units_Started And ProgrammingComplete Then

            Dim Cyclone3_GetResult As UShort
            Dim Cyclone4_GetResult As UShort

            Try
                '**** If Cyclone Status remains at 1; after an extended amount of time, the
module is presumed to be unable to program
                '**** In this case, set the result to 1 <Fail>

                If Cyclone3_Status <> 1 Then
                    Threading.Thread.Sleep(0)
                    Cyclone3_GetResult =
get_last_error_code(cyclonepromaxhandle:=Cyclone3_Handle)
                Else : Cyclone3_GetResult = 2  'represents a condition where the cyclone
has timed out; is still giving busy but is assuming fail condition
                End If

                If Cyclone4_Status <> 1 Then
                    Threading.Thread.Sleep(0)
```

```
                    Cyclone4_GetResult =
get_last_error_code(cyclonepromaxhandle:=Cyclone4_Handle)
                Else : Cyclone4_GetResult = 2  'represents a condition where the cyclone
has timed out; is still giving busy but is assuming fail condition
                End If

                e.Result = CStr(Cyclone3_GetResult) + CStr(Cyclone4_GetResult)  '0 =
pass,  1 = fail

            Catch ex As Exception
            End Try

        End If


        If bckgrndWrk_Ln2.CancellationPending = True Then
            e.Cancel = True
        End If

    End Sub

    Private Sub bckgrndWrk_Ln2_ProgressChanged(ByVal sender As System.Object,
ByVal e As System.ComponentModel.ProgressChangedEventArgs) Handles
bckgrndWrk_Ln2.ProgressChanged

        btn_upload_cyclone.Enabled = False 'disable uploading button during a
programming cycle
        btn_file_select.Enabled = False 'disable file selection while programming

        tempCount2 = tempCount2 + 1
        '  tempLabel_2.Text = CStr(tempCount2)
```

```
If dm_ID_3.Text <> dmCode_Ln2 Then
   dm_ID_3.Visible = True
   dm_ID_3.Text = dmCode_Ln2.Trim

End If

If dm_ID_4.Text <> dmCode_Ln2_2 Then
   dm_ID_4.Visible = True
   dm_ID_4.Text = dmCode_Ln2_2.Trim

End If


If prg_bar3.Value >= 95 Then 'Update the progress bar if it hasn't maxed out
   prg_bar3.Value = 95

Else : prg_bar3.Value = e.ProgressPercentage
End If


If prg_bar4.Value >= 95 Then 'Update the progress bar if it hasn't maxed out
   prg_bar4.Value = 95

Else : prg_bar4.Value = e.ProgressPercentage
End If

'Determine Cyclone Activity & Status
Dim status As String = CStr(e.UserState)
```

```
        If status = "start" Then
            Dim startChk As Boolean
            While Not startChk
                startChk = Write_ProServer(False, 0, False, 0, False, 0, False, 0, False, 0,
False, 0, True, 1) 'Update ProServer Bits <M909; Lane2 programming started
                If Not startChk Then
                    System.Threading.Thread.Sleep(50)
                End If

            End While

            lbl_status_Cyc3.Text = "Erasing"
            lbl_status_Cyc4.Text = "Erasing"

        ElseIf status = "busy" Then
            lbl_status_Cyc3.Text = "Programming"
            lbl_status_Cyc4.Text = "Programming"
        End If

    End Sub

    Private Sub bckgrndWrk_Ln2_RunWorkerCompleted(ByVal sender As
System.Object, ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs)
Handles bckgrndWrk_Ln2.RunWorkerCompleted

        Dim Lane2_Results As Boolean
        Dim resultsChk As Boolean
        Dim result As String
        Dim Cyc3_result, Cyc4_result As Integer
        Dim Cyc3_result_db, Cyc4_result_db As Char
        Cyc3_result = -1
```

```
            Cyc4_result = -1

            Dim boolReset3 As Boolean
            Dim boolReset4 As Boolean

            If e.Cancelled <> True Then
               result = e.Result.ToString  'get results for cyclones 3 & 4
               Cyc3_result = CInt(result.Substring(0, 1))
               Cyc4_result = CInt(result.Substring(1, 1))

               If Cyc3_result = 0 And Cyc4_result = 0 Then 'If both units report 0 then lane
has passed
                  Lane2_Results = True

               Else
                  If Cyc3_result = 2 Then      'If value is 2; cyclone unit requires a reset
command
                     boolReset3 = True
                  End If

                  If Cyc4_result = 2 Then       'If value is 2; cyclone unit requires a reset
command
                     boolReset4 = True
                  End If

               End If

            End If

            'Handle GUI services
            Select Case Cyc3_result
```

```
    Case 0
        lbl_status_Cyc3.Text = "Pass"
        Cyc3_result_db = CChar("P")
        ovl_inprogress_Cyc3.FillColor = Color.LightGreen
        prg_bar3.Value = 100

    Case 1
        lbl_status_Cyc3.Text = "FAIL"
        Cyc3_result_db = CChar("F")
        ovl_inprogress_Cyc3.FillColor = Color.Red
    Case Else
        lbl_status_Cyc3.Text = "FAIL"
        Cyc3_result_db = CChar("F")
        ovl_inprogress_Cyc3.FillColor = Color.Red
End Select

Select Case Cyc4_result

    Case 0
        lbl_status_Cyc4.Text = "Pass"
        Cyc4_result_db = CChar("P")
        ovl_Inprogress_Cyc4.FillColor = Color.LightGreen
        prg_bar4.Value = 100
    Case 1
        lbl_status_Cyc4.Text = "FAIL"
        Cyc4_result_db = CChar("F")
        ovl_Inprogress_Cyc4.FillColor = Color.Red
    Case Else
        lbl_status_Cyc4.Text = "FAIL"
        Cyc4_result_db = CChar("F")
```

```
            ovl_Inprogress_Cyc4.FillColor = Color.Red

        End Select


        Select Case Lane2_Results

          'Provide resutls to HMI via ProServer
          Case True
            While resultsChk <> True

              resultsChk = Write_ProServer(False, 0, True, 1, False, 0, False, 0, False,
0, False, 0, False, 0) 'Update ProServer Bits <M903; Lane2 passed programming

              If Not resultsChk Then
                System.Threading.Thread.Sleep(50)
              End If

            End While

            ovl_lane2.FillColor = Color.LightGreen
            lbl_results_lane2.Text = "Pass"

          Case False
            resultsChk = Write_ProServer(False, 0, False, 0, False, 0, False, 0, True, 1,
False, 0, False, 0) 'Update ProServer Bits <M906; Lane2 failed programming
            ovl_lane2.FillColor = Color.Red
            lbl_results_lane2.Text = "Fail"
        End Select
```

```
            Call Db_Services(dmCode_Ln2, 3, Cyc3_result_db)  'Handle db services for the
results from Lane 2 / PCB 1
            Call Db_Services(dmCode_Ln2_2, 4, Cyc4_result_db) 'Handle db services for
the results from Lane 2 / PCB 2

            'If necessary reset Cyclone Units
            If boolReset3 Or boolReset4 Then

               Call Reset_Cyclones_Lane1(boolReset3, boolReset4)
               Threading.Thread.Sleep(1000) 'sleep the thead to allow the Cyclone Units time
to reboot

            End If

            If bckgrndWrk_Ln1.IsBusy = False Then

               btn_file_select.Enabled = True 'enable file selection while programming

            End If

            If bckgrndWrk_ProSrv_Ln2.IsBusy <> True Then
               Call bckgrndWrk_ProSrv_Ln2.RunWorkerAsync() 'Restart the thread to
monitor Pro-Server bits for Lane 2
            End If


            tempCount2 = 0

         End Sub
         Private Function Db_Services(ByVal datamatrix As String, ByVal Unit_ID As
Integer, ByVal Result As Char) As Boolean
```

```
'***** Handle calls to perform database transactions *******
Dim intImageReturn, intBarcodeReturn As Integer
Dim blnImgRecord, blnBrcdRecord, blnBadRtrn As Boolean
Dim imageFile As String
imageFile = String.Empty

'Obtain the current Cyclone Image file being utilized
Select Case Unit_ID

  Case 1
    imageFile = Cyclone1_Image_Description
  Case 2
    imageFile = Cyclone2_Image_Description
  Case 3
    imageFile = Cyclone3_Image_Description
  Case 4
    imageFile = Cyclone4_Image_Description
End Select

'Get the image file & barcode string
intImageReturn = Check_Image_File(imageFile) 'Use stored procedure to query db for an existing record of the SAP image file in use
' A return of 1 indicates a matching record, a return of 0 indicates no matching record.
intBarcodeReturn = Check_Barcode(datamatrix) 'Use stored procedure to query db for an existing record of the barcode (serial #) in use
' A return of 1 indicates a matching record, a return of 0 indicates no matching record.

'Determine result of stored procedure calls
```

```
Select Case intImageReturn
  Case 0
    blnImgRecord = True
  Case 1
    blnImgRecord = False
  Case 255
    blnBadRtrn = False
End Select

Select Case intBarcodeReturn
  Case 0
    blnBrcdRecord = True
  Case 1
    blnBrcdRecord = False
  Case 255
    blnBadRtrn = False
End Select

If Not blnBadRtrn Then
  Dim dbUpdate As Integer = 0
  'Call function to transact with the db / updating the tables via the stored
procedure
  dbUpdate = Update_database(blnBrcdRecord, blnImgRecord, datamatrix,
imageFile, Unit_ID, Result)

  If Not (dbUpdate >= 1 And dbUpdate <= 2) Then
    Call Local_bck_file(datamatrix, imageFile, Unit_ID, Result) 'In the event of
a db connection problem, the data is backed up locally

  End If
Else
```

```
            Call Local_bck_file(datamatrix, imageFile, Unit_ID, Result) 'In the event of a
db connection problem, the data is backed up locally

            End If

            Return True
        End Function
        Private Function data_Cleanse() As Boolean 'Performs actions to retrieve locally
stored backup files and update to the db
            '************ Called upon form loading, via toolstrip user selection, & once
an auto-cycle is stopped by the user.

            Do While StrmRead()
              'Call to transact the local data storage files & uploda to the SQL db. As long as
files are being processed, will return true.
            Loop

            data_Cleanse = True
            Return data_Cleanse

        End Function
        Private Sub Lane2PassBitHighToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Lane2PassBitHighToolStripMenuItem.Click


            Write_ProServer(False, 0, True, 1, False, 0, False, 0, False, 0, False, 0, False, 0)
'Update ProServer Bits <M903; Lane2 passed programming


        End Sub
```

```
        Private Sub Lane2PassBitLowToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Lane2PassBitLowToolStripMenuItem.Click

            Write_ProServer(False, 0, True, 0, False, 0, False, 0, False, 0, False, 0, False, 0)
'Update ProServer Bits <M903; Lane2 passed programming


        End Sub

        Private Sub Lane1PassBitToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Lane1PassBitToolStripMenuItem.Click

            Write_ProServer(False, 0, False, 0, True, 1, False, 0, False, 0, False, 0, False, 0)
'Update ProServer Bits <M904; Lane1 passed programming

        End Sub

        Private Sub Lane2PassBitToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Lane2PassBitToolStripMenuItem.Click
            Write_ProServer(False, 0, False, 0, True, 0, False, 0, False, 0, False, 0, False, 0)
'Update ProServer Bits <M904; Lane1 passed programming

        End Sub

        Private Sub DataCleansingToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
DataCleansingToolStripMenuItem.Click
```

```
        DataCleansingToolStripMenuItem.Enabled = False
        Call data_Cleanse()
        DataCleansingToolStripMenuItem.Enabled = True

    End Sub
End Class
```

Cyclone.vb

```
Module Cyclone
    '***** API Decleratoins for Cyclone Pro Flashing Units
    '***** Tilden May 03/2013

    Declare Sub enumerate_all_ports Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" ()
    Declare Function connect_to_cyclonepromax Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal port_type As UInteger,
ByVal identifier_type As UInteger, ByVal port_identifier As String) As UInteger
    Declare Function get_image_description Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger, ByVal image_id As Byte) As String
    Declare Function START_execute_all_commands Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger, ByVal image_id As Byte) As Boolean
    Declare Function check_STARTED_cyclonepromax_status Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger) As UInteger
```

```
        Declare Function get_last_error_code Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger) As UInt16
        Declare Function reset_cyclonepromax Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger, ByVal reset_delay_in_ms As UInteger) As Boolean
        Declare Function compare_image_with_file Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger, ByVal aFile As String, ByVal image_id As Byte) As Boolean
        Declare Function erase_all_cyclone_images Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger) As Boolean
        Declare Function add_image_to_cyclone Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger, ByVal aFile As String) As UInteger
        Declare Function disconnect_from_cyclonepromax Lib
"c:\windows\system\CYCLONE_CONTROL_stdcall.dll" (ByVal cyclonepromaxhandle As
UInteger) As Boolean

        Public Cyclone1_Handle As UInteger
        Public Cyclone2_Handle As UInteger
        Public Cyclone3_Handle As UInteger
        Public Cyclone4_Handle As UInteger

        Public Const Cyclone1_IP As String = "172.19.131.212"
        Public Const Cyclone2_IP As String = "172.19.131.213"
        Public Const Cyclone3_IP As String = "172.19.131.214"
        Public Const Cyclone4_IP As String = "172.19.131.215"

        Public Sub Disconnect_Cyclones_Lane1()
```

```
    Dim boolTerm_1 As Boolean
    Dim boolTerm_2 As Boolean

    boolTerm_1 = disconnect_from_cyclonepromax(Cyclone1_Handle)
    boolTerm_2 = disconnect_from_cyclonepromax(Cyclone2_Handle)


End Sub

Public Sub Disconnect_Cyclones_Lane2()

    Dim boolTerm_3 As Boolean
    Dim boolTerm_4 As Boolean

    boolTerm_3 = disconnect_from_cyclonepromax(Cyclone3_Handle)
    boolTerm_4 = disconnect_from_cyclonepromax(Cyclone4_Handle)

End Sub


Public Sub Reset_Cyclones_Lane1(ByVal Cyclone1 As Boolean, ByVal Cyclone2
As Boolean)

    If Cyclone1 Then
      reset_cyclonepromax(Cyclone1_Handle, 2500)
    End If

    If Cyclone2 Then
      reset_cyclonepromax(Cyclone2_Handle, 2500)
    End If
End Sub
```

```
    Public Sub Reset_Cyclones_Lane2(ByVal Cyclone3 As Boolean, ByVal Cyclone4
As Boolean)

        If Cyclone3 Then
            reset_cyclonepromax(Cyclone3_Handle, 2500)
        End If

        If Cyclone4 Then
            reset_cyclonepromax(Cyclone4_Handle, 2500)
        End If


    End Sub
End Module
```

Database.vb

```
Option Explicit On
Option Strict On

Imports System.Data.SqlClient
Imports System.IO
```

```
Module database

'******* Handles database transactions including calling stored procedures on HK-ELECTRON-1. DB -> FA5.
'******* Also handles processing of local backup files in the event of a db connection / network error.
'******* A log file is generated when an attempt to transact the the db fails.
'******* Local data files will periodically be accessed and an attempt to update the db with the data wil occur.  If this is successful the local data files are then deleted.
'******* Developed by Tilden May 05/08/2013 **************

Public recordArray(1000, 3) As String 'array to temp hold the data
'*** Directory for storing local backup files ***
Private Const path = "C:\Prg_Data\db\Record\"
'*** Stored Procedure Names ***
Private Const ImageCheck = "ImageCheck"
Private Const barcodeCheck = "barcodeCheck"
Private Const Update_db = "Update_db"
'******************************
Private Const DCS = "Data Source=172.19.132.1; Initial Catalog=FA5;Persist Security Info = True;User ID=VS1;Password=HondaVS1"

Public Function Check_Image_File(ByVal ImageFile As String) As Integer
  '*** Call Stored Procedure to determine if a record of the image file exists in the FA5 database.
  '*** A return of 1 indicates a matching record, a return of 0 indicates no matching record.

  Dim connection As SqlConnection = New SqlConnection(DCS)
  Dim cmd As SqlCommand = New SqlCommand(ImageCheck, connection) '"Procedure Name", Connection Credentials
```

```
            Dim result_Return As Integer
            result_Return = 255 'Preload the integer

            cmd.CommandType = CommandType.StoredProcedure
            cmd.Parameters.AddWithValue("@image_file", ImageFile)
            cmd.Parameters.AddWithValue("@indicatorInt", result_Return)
            cmd.Parameters("@indicatorInt").Direction = ParameterDirection.InputOutput
'The parameter is both passed & returned

            Try
               'Execute the stored procedure
               connection.Open()
               cmd.ExecuteNonQuery()
               connection.Close()
               result_Return = CInt(cmd.Parameters("@indicatorInt").Value) 'Get the
returned response from executing the stored procedure.

            Catch ex As Exception
               Dim errorMessage As String
               errorMessage = ex.Message.ToString
               Call WriteLog(ImageCheck, errorMessage) 'The procedure type along with the
handling error is logged for reference
            Finally
               If connection.State = ConnectionState.Open Then
                  connection.Close()
               End If
            End Try

            Return result_Return
```

```
End Function

Public Function Check_Barcode(ByVal Barcode As String) As Integer
    '*** Call Stored Procedure to determine if a record of the barcode exists in the FA5 database.
    '*** A return of 1 indicates a matching record, a return of 0 indicates no matching record.

    Dim connection As SqlConnection = New SqlConnection(DCS)
    Dim cmd As SqlCommand = New SqlCommand(barcodeCheck, connection) '"Procedure Name", Connection Credentials

    Dim result_Return As Integer
    result_Return = 255 'Preload the integer

    cmd.CommandType = CommandType.StoredProcedure
    cmd.Parameters.AddWithValue("@barcode", Barcode)
    cmd.Parameters.AddWithValue("@indicatorInt", result_Return)
    cmd.Parameters("@indicatorInt").Direction = ParameterDirection.InputOutput 'The parameter is both passed & returned

    Try
      'Execute the stored procedure
      connection.Open()
      cmd.ExecuteNonQuery()
      connection.Close()
      result_Return = CInt(cmd.Parameters("@indicatorInt").Value) 'Get the returned response from executing the stored procedure.

    Catch ex As Exception
      Dim errorMessage As String
```

```
            errorMessage = ex.Message.ToString
            Call WriteLog(barcodeCheck, errorMessage) 'The procedure type along with
the handling error is logged for reference

        Finally
            If connection.State = ConnectionState.Open Then
                connection.Close()
            End If
        End Try

        Return result_Return

    End Function

    Public Function Update_database(ByVal blnBarcodeRecord As Boolean, ByVal
blnImageRecord As Boolean, ByVal barcode As String, ByVal ImageFile As String, ByVal
UnitID As Integer, ByVal prgResult As Char) As Integer
        '*** Call Stored Procedure to Insert a record into the FA5 db (Tables updated are
[*Image_File, *Product_Serial, & Prodcution_Results  *IF applicable]
        '***

        Dim connection As SqlConnection = New SqlConnection(DCS)
        Dim cmd As SqlCommand = New SqlCommand(Update_db, connection)
'"Procedure Name", Connection Credentials

        Dim result_Return As Integer
        result_Return = 255 'Preload the integer

        cmd.CommandType = CommandType.StoredProcedure
        cmd.Parameters.AddWithValue("@bln_new_barcode_record",
blnBarcodeRecord) 'indicates existing record status for the barcode
```

```
            cmd.Parameters.AddWithValue("@bln_new_image_file", blnImageRecord)
'indicates existing record status for the SAP file (image)
            cmd.Parameters.AddWithValue("@barcode", barcode) 'product barcode being
programmed
            cmd.Parameters.AddWithValue("@image", ImageFile) 'file name of SAP image
            cmd.Parameters.AddWithValue("@unit_ID", UnitID) 'indicates which lane, and
which Cyclone unit (composite) the product was functionally programmed on
            cmd.Parameters.AddWithValue("@result", prgResult) 'P = Pass / F = Fail
(Results of Programming Attempt)

            Try
               'Execute the stored procedure
               connection.Open()
               result_Return = cmd.ExecuteNonQuery()
               connection.Close()

            Catch ex As Exception
               Dim errorMessage As String
               errorMessage = ex.Message.ToString
               Call WriteLog(Update_db, errorMessage) 'The procedure type along with the
handling error is logged for reference
            Finally
               If connection.State = ConnectionState.Open Then
                  connection.Close()
               End If
            End Try

            Return result_Return

         End Function
```

```
Public Function WriteLog(ByRef ProcedureCall As String, ByVal errorMessage As
String) As Boolean 'Write Error Log for any db connection issues

    Dim strMyTime As String
    Dim LogLocation As String = "C:\Prg_Data\db\Log\db_log" + "_" +
Date.Today.ToLongDateString() + ".log" 'Directory for storing SQL connection error log files
    Dim msg As String = "Attempted Call To"
    Dim fs As System.IO.FileStream
    Dim fw As System.IO.StreamWriter

    strMyTime = System.DateTime.Now.Hour.ToString.PadLeft(2,
Convert.ToChar("0")) + ":" + System.DateTime.Now.Minute.ToString.PadLeft(2,
Convert.ToChar("0")) + ":" + System.DateTime.Now.Second.ToString.PadLeft(2,
Convert.ToChar("0"))

    fs = New FileStream(LogLocation, FileMode.Append, FileAccess.Write)
    fw = New StreamWriter(fs)

    Try

      fw.WriteLine()
      fw.WriteLine(msg + " " + ProcedureCall + " " + errorMessage + "  " +
strMyTime)
      fw.Close()
      fs.Close()

    Catch ex As Exception

      Dim err As String
      err = ex.Message.ToString
```

```
            WriteLog = False
            fw.Close()
            fs.Close()
            Exit Function

        End Try

        WriteLog = True

    End Function

    Public Function Local_bck_file(ByVal barcode As String, ByVal ImageFile As
String, ByVal UnitID As Integer, ByVal prgResult As Char) As Boolean

        '******* Write backup data file to local drive in the event the database / network
connection is not available

        Dim strMyTime As String
        Dim pthBarcode As String

        strMyTime = System.DateTime.Now.Hour.ToString.PadLeft(2,
Convert.ToChar("0")) & ":" & System.DateTime.Now.Minute.ToString.PadLeft(2,
Convert.ToChar("0")) & ":" & System.DateTime.Now.Second.ToString.PadLeft(2,
Convert.ToChar("0"))
        pthBarcode = barcode.Remove(0, 14)

        'Dim temp As String
        'temp = "GE123456A011301019876"

        Dim RecordLocation As String = "C:\Prg_Data\db\Record\" + pthBarcode + "_"
+ Date.Today.ToLongDateString() + ".log" ' Date.Today.ToLongDateString() & ".log"
```

```
        Dim fs As System.IO.FileStream
        Dim fw As System.IO.StreamWriter


        Try
            fs = New FileStream(RecordLocation, FileMode.Append, FileAccess.Write)
'Create New or Append to an existing file as needed
            fw = New StreamWriter(fs)

            fw.WriteLine(barcode & ", " & ImageFile & ", " & UnitID & ", " & prgResult
& ", ....." & strMyTime)
            fw.Close()

        Catch ex As Exception
            Dim errorMessage As String
            errorMessage = ex.Message.ToString
            Local_bck_file = False
            Exit Function
        End Try

        Local_bck_file = True


    End Function
    Public Function StrmRead() As Boolean 'Function to Open / Retrieve Local Stored
Data In Preperation For DB Updating.  Each function cycle will open / read one file &
populate data into an array.  (*if files are present)

        StrmRead = False
```

```
' make a reference to a directory
Dim di As New IO.DirectoryInfo(path)
Dim diar1 As IO.FileInfo() = di.GetFiles()
Dim fn As IO.FileInfo

If diar1.Length <> 0 Then 'Bounds inidcates presence of files
   StrmRead = True
   'Obtain the data files for processing
   fn = diar1(0) 'Get first file stored in first array element

   '**************** Clear all values from array elements
**********************
   Array.Clear(recordArray, recordArray.GetLowerBound(0),
recordArray.Length)
   Dim z As Integer
   z = 0

   Dim RecordLocation As String = path & fn.ToString 'Set file name to string
variable
   '***************** Handle File Access / Open / Read
************************
   Dim rs As System.IO.FileStream
   Dim rr As System.IO.StreamReader

   rs = New FileStream(RecordLocation, FileMode.Open, FileAccess.Read)
'Specifiy file name, method (open) and access level (read)
   rr = New StreamReader(rs)

   While rr.Peek <> -1  'Loop unitl EOF

     Dim fullLine, barcode, SAP, Unit, Result, tempHandle As String
```

```
fullLine = String.Empty
barcode = String.Empty
SAP = String.Empty
Unit = String.Empty
Result = String.Empty
Dim index, cycleCount As Integer

cycleCount = 4 'Load counter, will decrement as loop counter

fullLine = rr.ReadLine() 'Read Line from Record Text File
Do Until cycleCount <= 0

  index = fullLine.IndexOf(",", 0) 'Index location of post-text comma
  tempHandle = fullLine.Substring(0, index) 'Extract the barcode portion of
the string using x to identify length
  fullLine = fullLine.Remove(0, index + 2) 'Truncate the reaminder of the
complete string by removing the barcode portion (including comma + space)

  'Populate the variables through the loop iterarations
  Select Case cycleCount

    Case 4
      barcode = tempHandle
    Case 3
      SAP = tempHandle
    Case 2
      Unit = tempHandle
    Case 1
      Result = tempHandle

  End Select
```

```
                cycleCount = cycleCount - 1 'decrement loop counter

            Loop
            'array storage of file data
            recordArray(z, 0) = barcode
            recordArray(z, 1) = SAP
            recordArray(z, 2) = Unit
            recordArray(z, 3) = Result

            z = z + 1 'increment the array index counter

        End While

        rr.Close() 'Close file access

        StrmRead = Process_local_data(fn.ToString) 'Function will perform the
transactions to update the db with the stored local data.
        Else : StrmRead = False 'In the event of no files present or remaining in the
directoy

        End If

        Return StrmRead

    End Function
    Private Function Process_local_data(ByVal fileName As String) As Boolean
'********** Uses acquired local data to transact with the db.  If the  db transaction is
successfful then delete the local storage file.  **********

        Dim success As Boolean
```

```
Dim Getbarcode, GetSAP, GetUnit, GetResult As String
Dim a As Integer 'array index counter
a = 0

Do Until IsNothing(recordArray(a, 0)) 'Loop through array elements while data is present.  Each cycle iteration will transact the stored data -> SQL db.

    'Clear the holder variables
    Getbarcode = String.Empty
    GetSAP = String.Empty
    GetResult = String.Empty
    GetUnit = String.Empty

    'Access row data from multi-demensional array
    Getbarcode = recordArray(a, 0)
    GetSAP = recordArray(a, 1)
    GetUnit = recordArray(a, 2)
    GetResult = recordArray(a, 3)

    '*****  Perform db stored procedure calls *****
    Dim intCIF, intCB, intUpdb, Unit As Integer
    Dim Result As Char
    Dim blnSAP, blnBarcode As Boolean

    'Call function to run Check Image stored procedure
    intCIF = Check_Image_File(GetSAP)  '255 = db procedure did not return a value.  1 = record found, 0 = no record found.
    If intCIF <> 255 Then
       Select Case intCIF
          Case 0 'Case no record present in the Image_File table; the image file needs to be inserted
```

```
                blnSAP = True
            Case 1 'Case A record is present in the Image_File table; the image file
does not need to be inserted
                blnSAP = False
        End Select

        'Call function to run Check Barcode stored procedure
        intCB = Check_Barcode(Getbarcode)
        If intCB <> 255 Then
            Select Case intCB
                Case 0  'Case no record present in the Product_Serial table; the barcode
needs to be inserted
                    blnBarcode = True
                Case 1   'Case A record is present in the Product_Serial table; the
barcode does not need to be inserted
                    blnBarcode = False
            End Select

            'Data Type Conversion
            Try
                If IsNumeric(GetUnit) Then
                    Unit = Convert.ToInt16(GetUnit)
                End If
                Result = Convert.ToChar(GetResult)

            Catch ex As Exception
                success = False
            End Try

            Try
                'Call function to run Update Database stored procedure
```

```
                    intUpdb = Update_database(blnBarcode, blnSAP, Getbarcode,
GetSAP, Unit, Result)
                    success = True
                    '*** not sure of the expected returned value yet ?? *********
                    'If intUpdb = 1 Or intUpdb = 2 Then
                    '   success = True
                    'End If
                  Catch ex As Exception
                    success = False
                  End Try
                Else : success = False

                End If
              Else : success = False

              End If

              a = a + 1 'increment the array index counter
            Loop

            ' success = True
            If success Then

              '**** Once the data has SQL db has been updated the local storage file can be
deleted.  Erase record *********
              Dim di As New IO.DirectoryInfo(path)
              Dim diar1 As IO.FileInfo() = di.GetFiles()
              Dim fn As IO.FileInfo

              For Each fn In diar1
```

```
                If fn.ToString = fileName Then 'delete the current file that had been
processed.
                    fn.Delete()
                    Exit For

                End If
            Next

        End If

        Return success

    End Function

    Private Sub FileStream(ByVal p1 As Object)
        Throw New NotImplementedException
    End Sub

    Private Sub Stream(ByVal p1 As Object)
        Throw New NotImplementedException
    End Sub

End Module
```

ProServer_Exchange.vb

```
    Module ProServer_Exchange
        'Functions to Read / Write from Application -> Machine Side Controls via
ProServer API
        'Tilden May 03/2013
```

```
Const setup As String = "12345678901234567890123456789O"
Public dmCode_Ln1_2 As String 'hold the datamatrix ID code for product processed on lane 1 / board 2  **PCB 1 of panel is actually code scanned / received
Public dmCode_Ln2_2 As String 'hold the datamatrix ID code for product processed on lane 2 / board 2  **PCB 1 of panel is actually code scanned / received
Public dmCode_Ln1 As String 'hold the datamatrix ID code for product processed on lane 1 / board 1  **PCB 1 of panel is actually code scanned / received
Public dmCode_Ln2 As String 'hold the datamatrix ID code for product processed on lane 2 / board 1 **PCB 1 of panel is actually code scanned / received

Public Function Write_ProServer(ByVal M900 As Boolean, ByVal M900_i As Integer, ByVal M903 As Boolean, ByVal M903_i As Integer, ByVal M904 As Boolean, ByVal M904_i As Integer, ByVal M905 As Boolean, ByVal M905_i As Integer, ByVal M906 As Boolean, ByVal M906_i As Integer, ByVal M909 As Boolean, ByVal M909_i As Integer, ByVal M910 As Boolean, ByVal M910_i As Integer) As Boolean

    Dim ProServer_Comm As Boolean = True

    'M900 -- Application is Active
    'M903 -- Lane 1 Pass
    'M903 -- Lane 2 Pass
    'M905 -- Lane 1 Fail
    'M906 -- Lane 2 Fail
    'M909 -- Lane 1 started
    'M910 -- Lane 2 started

    Try

      If M900 Then
        WriteDeviceBit("Proface.PLC1", "M900", M900_i, 1)
      End If
```

```
    If M903 Then
        WriteDeviceBit("Proface.PLC1", "M903", M903_i, 1)
    End If

    If M904 Then
        WriteDeviceBit("Proface.PLC1", "M904", M904_i, 1)
    End If

    If M905 Then
        WriteDeviceBit("Proface.PLC1", "M905", M905_i, 1)
    End If

    If M906 Then
        WriteDeviceBit("Proface.PLC1", "M906", M906_i, 1)
    End If

    If M909 Then
        WriteDeviceBit("Proface.PLC1", "M909", M909_i, 1)
    End If

    If M910 Then
        WriteDeviceBit("Proface.PLC1", "M910", M910_i, 1)
    End If

Catch ex As Exception
    Dim errorMessage As String
    errorMessage = ex.Message.ToString
    ProServer_Comm = False
End Try
```

```
    Return ProServer_Comm

End Function

Public Function Read_ProServer(ByVal Lane_Request As Integer) As Integer

    Dim PrServ_Request_Strt As Integer
    Dim monitorBit As Integer = 0
    Dim blnLn1ProjectedID As Boolean
    Dim blnLn2ProjectedID As Boolean


    Try
      Select Case Lane_Request

        Case 1 'Monitor Pro Server bit for lane 1
          dmCode_Ln1 = setup
          ReadDeviceBit("Proface.PLC1", "M901", monitorBit, 1)  'M901 - request
to start lane bit from HMI (API)
          ReadDeviceStr("Proface.PLC1", "WR100", dmCode_Ln1, setup.Length)
'WR100 - datamatrix barcode sent from HMI (API)

          If dmCode_Ln1 <> setup Then
            dmCode_Ln1.Trim()
            dmCode_Ln1 = dmCode_Ln1.Substring(0, 29)
            blnLn1ProjectedID = project_boardID(Lane_Request, dmCode_Ln1)
          End If

          If monitorBit > 0 And dmCode_Ln1 <> setup And blnLn1ProjectedID
Then
            PrServ_Request_Strt = 1
```

```
                End If



            Case 2  'Monitor Pro Server bit for lane 2
                dmCode_Ln2 = setup
                ReadDeviceBit("Proface.PLC1", "M902", monitorBit, 1)  'M902 - request
to start lane bit from HMI (API)
                ReadDeviceStr("Proface.PLC1", "WR200", dmCode_Ln2, setup.Length)
'WR200 - datamatrix barcode sent from HMI (API)

                If dmCode_Ln2 <> setup Then
                  dmCode_Ln2.Trim()
                  dmCode_Ln2 = dmCode_Ln2.Substring(0, 29)
                  blnLn2ProjectedID = project_boardID(Lane_Request, dmCode_Ln2)
                End If

                If monitorBit > 0 And dmCode_Ln2 <> setup And blnLn2ProjectedID
Then
                  PrServ_Request_Strt = 1
                End If
            End Select

        Catch ex As Exception
            Dim errorMessage As String
            errorMessage = ex.Message.ToString

        End Try

        Return PrServ_Request_Strt
```

```
End Function
Private Function project_boardID(ByVal lane As Integer, ByVal ID_code As
String) As Boolean 'Handle ID processing for 2nd pcb identity

    'Get the serial # from the scanned board ID
    Dim serial As Integer
    Dim ID_code_serial As String = String.Empty
    project_boardID = False

    If ID_code.Length >= 24 Then
      Try
        ID_code_serial = ID_code.Substring(25, 4)
        ID_code_serial.Trim()
      Catch ex As Exception
        Dim errorMessage As String
        errorMessage = ex.Message.ToString
      End Try


      If IsNumeric(ID_code_serial) Then
        serial = CInt(ID_code_serial)

        'Determine the serial of the secondary pcb.  Decrement the count by 1 (-1)
** Count range is 1 ~ 9999.
        Dim supplement_serial As Integer
        If serial >= 2 And serial <= 9999 Then
          supplement_serial = serial - 1
        ElseIf serial = 1 Then
          supplement_serial = 9999
        End If
```

```
                'Concatenate projected serials # to DM code
                Select Case lane


                  Case 1
                    dmCode_Ln1_2 = String.Empty
                    dmCode_Ln1_2 = ID_code.Substring(0, 25) &
supplement_serial.ToString("D4")
                  Case 2
                    dmCode_Ln2_2 = String.Empty
                    dmCode_Ln2_2 = ID_code.Substring(0, 25) &
supplement_serial.ToString("D4")

                End Select

                project_boardID = True


              End If
            End If

            Return project_boardID

          End Function
        End Module
```

APPENDIX C: SQL STORED PROCEDURES

FA5_Create_Tables

```
Use FA5

/*
drop table Product_serial
 drop table Image_File
 drop table Cyclones
drop table Production_Results
*/

Create table Product_Serial
(serial_ID Int IDENTITY,
datamatrix_barcode varChar(40) not null Unique,
Primary Key (serial_ID)
);


Create table Image_File
(image_ID Int IDENTITY,
cyclone_image varChar(120) not null unique,
Primary key (image_ID)
);


Create table Cyclones
(unit_ID tinyInt,
lane tinyInt not null,
ip_address varChar(15) not null Unique,
Primary key (unit_id)
);
```

```
Create table Production_Results
(result_ID Int not null,
date_time DateTime2(0) not null,
unit tinyInt not null,
result varChar(1) not null,
cyclone_image int not null,
);

Alter table Production_Results
Add Constraint prod_results_PK Primary Key (result_ID, date_time);

Alter table Production_Results
Add Constraint result_ID_FK
Foreign Key (result_ID)
References Product_Serial (serial_ID);

Alter table Production_Results
Add Constraint unit_FK
Foreign Key (unit)
References Cyclones (unit_ID);

Alter table Production_Results
Add Constraint image_FK
Foreign Key (cyclone_image)
References Image_File (image_ID);
```

FA5_Create_ImageCheck_Procedure

```
--Procedure Creation for FA5 Programming DB Transactions / Tilden May /
02/28/2013

Use FA5
Go

--Drop Procedure ImageCheck

Create Procedure ImageCheck

        @image_file varChar (120),
        @indicatorInt tinyInt out -- variable that is returned; Used to indicate if a
current record exists for the given Cyclone Pro Image File

As
If @image_file in
        (Select cyclone_image from Image_File)  -- query db for record of product

        Begin
                Set @indicatorInt = 1
        End;

Else
        Begin
                Set @indicatorInt = 0
        End;
Return
Go
```

FA5_Create_barcodeCheck_Procedure

```
Use FA5
Go

-- Drop Procedure barcodeCheck

Create Procedure barcodeCheck

        @barcode varChar (40),
        @indicatorInt tinyInt out -- variable that is returned; Used to indicate if a
current record exists for the given barcode

As
If @barcode in
        (Select datamatrix_barcode from Product_Serial)  -- query db for record of
product

        Begin
                Set @indicatorInt = 1
        End;

Else
        Begin
                Set @indicatorInt = 0
        End;

Return

Go
```

FA5_Create_Update_db_Procedure

```
--Procedure Creation for FA5 Programming DB Transactions / Tilden May / 02/28/2013

Use FA5
Go

--Drop Procedure Update_db

Create Procedure Update_db

        @bln_new_barcode_record bit,  --indicates existing record status for the barcode
        @bln_new_image_file bit, --indicates existing record status for the SAP file (image)
        @barcode varChar(40), --product barcode being programmed
        @image varChar(120), --file name of SAP image
        @unit_ID tinyInt, --indicates which lane, and which Cyclone unit (composite) the product was functionally programmed on
        @result char(1) -- P = Pass, F = Fail

        As

If @bln_new_barcode_record = 1
        Begin
                -- The Product_serial table is only updated in the event of a new barcode record
                Insert Into Product_Serial (datamatrix_barcode)
                Values (@barcode) -- serial_ID will autofill / increment per each transaction
```

```
            End;

        If @bln_new_image_file = 1
            Begin
                -- The Image table is only updated in the event of a new programming
file (*SAP file as retracted from the Cyclone units during each model selection cycle)
                Insert Into Image_File (cyclone_image)
                Values (@image); -- iamge_ID will autofill / increment per each
transaction
            End;

            Begin

                Declare @get_ID bigInt
                Set @get_ID = (Select serial_ID  -- get the ID# of the barcode from the
Produt_serial table
                                From Product_serial
                                Where datamatrix_barcode = @barcode);

                Declare @get_image bigInt
                Set @get_image = (Select image_ID  -- get the ID# of the barcode from
the Produt_serial table
                                    From Image_File
                                    Where cyclone_image = @image);

            End;

        -- Insert into the Production_Results table once the supporting tables have been
updated

            Begin
```

```
                Insert Into Production_Results (result_ID, date_time, unit, result,
cyclone_image)
                Values (@get_ID, getdate(), @unit_ID, @result, @get_image);

        End;

    Return

    Go
```

FA5_Insert_Cyclones

```
Use FA5
Go


Insert Into Cyclones (unit_ID, lane, ip_address)
Values (1, 1, '172.19.131.212');

Insert Into Cyclones (unit_ID, lane, ip_address)
Values (2, 1, '172.19.131.213');

Insert Into Cyclones (unit_ID, lane, ip_address)
Values (3, 2, '172.19.131.214');

Insert Into Cyclones (unit_ID, lane, ip_address)
Values (4, 2, '172.19.131.215');
```

-- static data; this holds the unit information for each of the Cyclone Modules. Consists on an ID #, the lane it resides in, and the network address

Execute_barcodeCheck

```
Use FA5  -- Test TSQL query for testing 'Create_barcodeCheck_Procedure' Tilden May / 02/28/2013
Go

Declare          @barcode varChar (40)  --Delcare variables for passed paramaters
Declare @record_exists tinyInt    -- variable that is returned (0 ~ 255)

Set @barcode = 'GE123456A011301011234'
Set @record_exists = 255   -- preload the variable; check the return value to indicate status of record check


Execute barcodeCheck @barcode, @record_exists out

Select @record_exists -- 0 = no existing record, 1 = a record resides in the Image table
```

Execute_ImageCheck

```
Use FA5  -- Test TSQL query for testing 'Create_ImageCheck_Procedure'  Tilden May / 02/28/2013
Go

Declare          @image_file varChar (120)  --Delcare variables for passed paramaters
Declare @image_exists tinyInt    -- variable that is returned (0 ~ 255)

Set @image_file = 'test file 1 2 3 DLCM-220 SAP Cyclone yada yada yada'
Set @image_exists = 255   -- preload the variable; check the return value to indicate status of record check


Execute ImageCheck @image_file, @image_exists out

Select @image_exists -- 0 = no existing record, 1 = a record resides in the Image table
```

Execute_Update_db

```
Use FA5  -- Test TSQL query for testing 'Update_db Procedure'  Tilden May /
02/28/2013
Go
--Variable Declerations to Pass to Update_db Procedure
Declare @bln_record_bit bit
Declare @bln_image_bit bit
Declare        @barcode_test varChar (40)  --Delcare variables for passed paramaters
Declare @imageFile varChar(120)
Declare @unitTest tinyInt
Declare @result_Test char(1)    -- variable that is returned (0 ~ 255)
--Load the variables
Set @bln_record_bit = 0
Set @bln_image_bit = 0
Set @barcode_test = 'GE123456A011301011234'
Set @imageFile = 'test 1 2 3 DLCM-220 SAP Cyclone yada yada yada'
Set @unitTest = 3
Set @result_Test = 'P'

--Call procedure
Execute Update_db @bln_record_bit, @bln_image_bit, @barcode_test, @imageFile,
@unitTest, @result_Test

--Sample the record of the serial_ID
Declare @get_ID bigInt
                Set @get_ID = (Select serial_ID  -- get the ID# of the barcode from the
Produt_serial table
                                From Product_serial
                                Where datamatrix_barcode = @barcode_test);
Select @get_ID -- return ID indicates success of procedural update
```

APPENDIX D: WEBSITE SOURCE CODE

welcomeStruts.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>

<html:html lang="true">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title><bean:message key="welcome.title"/></title>
    <html:base/>
  </head>

  <body style=background-color:lightsteelblue;>

    <logic:notPresent name="org.apache.struts.action.MESSAGE"
scope="application">
      <div  style="color: blue">
        ERROR:  Application resources not loaded -- check servlet container
        logs for error messages.
      </div>
    </logic:notPresent>

    <h1 style="text-align: center"><bean:message key="welcome.heading"/></h1>
```

```
</br>
<h1 style="text-align: center"> <img src ="hitachi.gif" alt = "Hitach Logo" />
</h1> <!-- header displaying logo in center of web form -->

<form name="Get_Results" action="results_servlet" method="post">

</br>

<fieldset style="text-align: center">

<h2 style="text-align: center">< bean:message
key="welcome.select_2"/></h2>

</br>

<input name="barcode" size ="30" id="barcode_ID" value="" />

</fieldset>

<br>
<br>
<br>
<br>
<br>
<br>
<br>

<!-- <h2 style="text-align: center">< bean:message
key="welcome.select_1"/></h2> <!-- Submit Caption -->
```

```
                <fieldset style="text-align: center">

               <!--     <input type="hidden" name="select" value="" />  -->

                    <input type="image" src="submit.png" value="" name="submit"
onclick="this.form.select.value = this.form.barcode"/>

                </fieldset>

            </form>

                <br>
                <br>
                <br>
                <br>
                <br>
                <br>
                <br>
                <br>
                <br>
                <br>
                <br>
                <br>

            <address style="text-align: center">
                HIAMS(AM)-HK &bull; 955 Warwick Road  &bull; Harrodsburg, Ky 40330
&bull; 859.734.5491
            </address>

        </body>
    </html:html>
```

results_servlet.java

```
*
 *Client / Server servlet for data access to retrieve
 * results from FA5 DLCM microprocessor programmer
 *By Tilden May
 *05/7/2013
 *
 */

package Server;

// import com.mysql.jdbc.DatabaseMetaData;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;


/**
 *
 * @author Tilden
 */
@WebServlet(name = "results_servlet", urlPatterns = {"/results_servlet"})
```

```
public class results_servlet extends HttpServlet {

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        String getbarcode;  // declare variables for processing user input
        getbarcode = "";

        if ("barcode_ID" != null) {
            getbarcode = request.getParameter("barcode");

            //select is the input from the jsp page on client side to submit the form
          }

        //Establish SQL queries based on user input from client
//      String test_query = "select pcb_ID, pcb_code, pcb_result\n" +
//      "from NB_test_tbl where pcb_ID =" + "'" + getbarcode + "'";
```

```
        //***********SQL Server Query for providing results snapshot to
user*****************************
            String test_query = "Select date_time, result, Image_File.cyclone_image, lane,
unit_ID\n" +
        "from Production_Results, Image_File, Cyclones\n" +
        "where result_ID = (Select serial_ID from Product_Serial where datamatrix_barcode
=" + "'" + getbarcode + "')\n" +
        "And Production_Results.cyclone_image = image_ID\n" +
        "And unit = unit_ID\n" +
        "Order By date_time";
        //*************************************************************************
**************************

            String use_query = "";
            use_query = test_query;
            String image = "<img src='pcb assembly.jpg' alt='image' />";

            //html tags (prepare heading of page)

//*****************************************************************
            out.println("<html>");
            out.println("<body style=background-color:lightsteelblue;>");
            out.println("<fieldset style='text-align: center'>");
            out.println("<h1>" + "FA5 Programmer Results" + "</h1>");
            out.println("</hr>");
            out.println("</br>");
            out.print("<img src='hitachi.gif' alt='image' />");
            out.println("</br>");
            out.println("</br>");
            out.println("</br>");
            out.println("<UL>");
```

```
            out.println("<h2>" + getbarcode + "</h2>");
            out.println("</br>");

//*****************************************************************

            // Make a centered table to display the data results
            out.println("<CENTER>");
            out.println("<TABLE BORDER=4 CELLPADDING=0 CELLSPACING=0
WIDTH=80%>");

            // First row
            out.println("<TR>");
            out.println("<TD COLSPAN=2><CENTER><B>");
            out.println("<FONT SIZE=+1>Results</FONT>");
            out.println("</B></CENTER></TD>");

            out.println("<TD><CENTER>");
            out.println(image);
            out.println("</CENTER></TD>");

            out.println("<TD COLSPAN=2><CENTER><B><FONT SIZE=+1>");
            out.println("Machine Information");
            out.println("</FONT></B></CENTER></TD>");
            out.println("</TR>");

            // Second row
            out.println("<TR>");
            out.println("<TH><CENTER>");
            out.println("Date / Time");
            out.println("</CENTER></TH>");
```

```
out.println("<TH><CENTER>");
out.println("Result");
out.println("</CENTER></TH>");

out.println("<TH><CENTER>");
out.println("Cyclone Image");
out.println("</CENTER></TH>");

out.println("<TH><CENTER>");
out.println("Lane Processed");
out.println("</TH>");

out.println("<TH><CENTER>");
out.println("Cyclone Unit Number");
out.println("</CENTER></TH>");
out.println("</TR>");


try {  //establish a connection to the db and perform query

 //Test SQL db on local db engine*******************
 /*
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();
 String connectionUrl = "jdbc:sqlserver://localhost:1433;" +
 "databaseName=nbtest;user=test;password=Uk8titles;";
 Connection con = DriverManager.getConnection(connectionUrl);  */


Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();
 String connectionUrl = "jdbc:sqlserver://172.19.132.1:1433;" +
 "databaseName=FA5;user=VS1;password=HondaVS1;";
```

```
Connection con = DriverManager.getConnection(connectionUrl);

con.getMetaData();
Statement st = con.createStatement();
ResultSet myResult = st.executeQuery(use_query); //execute the applicable
query


//loop to populate the table with the query results

 while (myResult.next()) {

   //Add data rows to table for each iteration of the loop; results from SQL
query
   out.println("<TR>");
   out.println("<TD><CENTER>");
   out.println(myResult.getString(1));
   out.println("</CENTER></TD>");

   out.println("<TD><CENTER>");
   out.println(myResult.getString(2));
   out.println("</CENTER></TD>");

   out.println("<TD><CENTER>");
   out.println(myResult.getString(3));
   out.println("</CENTER></TD>");

   out.println("<TD><CENTER>");
   out.println(myResult.getString(4));
   out.println("</CENTER></TD>");
```

```
            out.println("<TD><CENTER>");
            out.println(myResult.getString(5));
            out.println("</CENTER></TD>");
            out.println("</TR>");


        }
        //close db connection streams
       myResult.close();
        st.close();
       con.close();


    } catch (Exception e) {
        String message;
        message = e.toString();

        out.println(message);

    }

    out.close(); //close the print writer

    out.println("</fieldset>");
    out.println("</body>");
    out.println("</html>");


}
```

```
    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the +
sign on the left to edit the code.">
    /**
     * Handles the HTTP
     * <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);


    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
```

```
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```